Python Machine Learning Case Studies

Five Case Studies for the Data Scientist

Danish Haroon

Apress®

Python Machine Learning Case Studies

Five Case Studies for the Data Scientist

Danish Haroon

Apress®

Python Machine Learning Case Studies

Danish Haroon
Karachi, Pakistan

ISBN-13 (pbk): 978-1-4842-2822-7 ISBN-13 (electronic): 978-1-4842-2823-4
DOI 10.1007/978-1-4842-2823-4

Library of Congress Control Number: 2017957234

Cover image by Freepik (www.freepik.com)

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Celestin Suresh John
Development Editor: Matthew Moodie
Technical Reviewer: Somil Asthana
Coordinating Editor: Sanchita Mandal
Copy Editor: Lori Jacobs
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-2822-7. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Contents at a Glance

Contents

About the Author



Danish Haroon currently leads the Data Sciences team at Market IQ Inc, a patented predictive analytics platform focused on providing actionable, real-time intelligence, culled from sentiment inflection points. He received his MBA from Karachi School for Business and Leadership, having served corporate clients and their data analytics requirements. Most recently, he led the data commercialization team at PredictifyME, a startup focused on providing predictive analytics for demand planning and real estate markets in the US market. His current research focuses on the amalgam of data sciences for improved customer experiences (CX).

About the Technical Reviewer



Somil Asthana has a BTech from IITBHU India and a MS from the University of New York at Buffalo (in the United States) both in Computer Science. He is an entrepreneur, machine learning wizard, and BigData specialist consulting with fortune 500 companies like Sprint, Verizon , HPE, and Avaya. He has a startup which provides BigData solutions and Data Strategies to Data Driven Industries in ecommerce, content/ media domain.

Acknowledgments

I would like to thank my parents and lovely wife for their continuous support throughout this enlightening journey.

Introduction

This volume embraces machine learning approaches and Python to enable automatic rendering of rich insights and solutions to business problems. The book uses a hands-on case study-based approach to crack real-world applications where machine learning concepts can provide a best fit. These smarter machines will enable your business processes to achieve efficiencies in minimal time and resources.

Python Machine Learning Case Studies walks you through a step-by-step approach to improve business processes and help you discover the pivotal points that frame corporate strategies. You will read about machine learning techniques that can provide support to your products and services. The book also highlights the pros and cons of each of these machine learning concepts to help you decide which one best suits your needs.

By taking a step-by-step approach to coding you will be able to understand the rationale behind model selection within the machine learning process. The book is equipped with practical examples and code snippets to ensure that you understand the data science approach for solving real-world problems.

Python Machine Leaarning Case Studies acts as an enabler for people from both technical and non-technical backgrounds to apply machine learning techniques to real-world problems. Each chapter starts with a case study that has a well-defined business problem. The chapters then proceed by incorporating storylines, and code snippets to decide on the most optimal solution. Exercises are laid out throughout the chapters to enable the hands-on practice of the concepts learned. Each chapter ends with a highlight of real-world applications to which the concepts learned can be applied. Following is a brief overview of the contents covered in each of the five chapters:

Chapter 1 covers the concepts of statistics and probability.

Chapter 2 talks about regression techniques and methods to fine-tune the model.

Chapter 3 exposes readers to time series models and covers the property of stationary in detail.

Chapter 4 uses clustering as an aid to segment the data for marketing purposes.

Chapter 5 talks about classification models and evaluation metrics to gauge the goodness of these models.

CHAPTER 1

Statistics and Probability

The purpose of this chapter is to instill in you the basic concepts of traditional statistics and probability. Certainly many of you might be wondering what it has to do with machine learning. Well, in order to apply a best fit model to your data, the most important prerequisite is for you to understand the data in the first place. This will enable you to find out distributions within data, measure the goodness of data, and run some basic tests to understand if some form of relationship exists between dependant and independent variables. Let's dive in.

Note This book incorporates **Python 2.7.11** as the de facto standard for coding examples. Moreover, you are required to have it installed it for the *Exercises* as well.

So why do I prefer Python 2.7.11 over Python 3x? Following are some of the reasons:

- Third-party library support for Python 2x is relatively better than support for Python 3x. This means that there are a considerable number of libraries in Python 2x that lack support in Python 3x.
- Some current Linux distributions and macOS provide Python 2x by default. The objective is to let readers, regardless of their OS version, apply the code examples on their systems, and thus this is the choice to go forward with.
- The above-mentioned facts are the reason why companies prefer to work with Python 2x or why they decide not to migrate their code base from Python 2x to Python 3x.

Case Study: Cycle Sharing Scheme—Determining Brand Persona

Nancy and Eric were assigned with the huge task of determining the brand persona for a new cycle share scheme. They had to present their results at this year's annual board meeting in order to lay out a strong marketing plan for reaching out to potential customers.

D. Haroon, *Python Machine Learning Case Studies*, DOI 10.1007/978-1-4842-2823-4_1

The cycle sharing scheme provides means for the people of the city to commute using a convenient, cheap, and green transportation alternative. The service has 500 bikes at 50 stations across Seattle. Each of the stations has a dock locking system (where all bikes are parked); kiosks (so customers can get a membership key or pay for a trip); and a helmet rental service. A person can choose between purchasing a membership key or short-term pass. A membership key entitles an annual membership, and the key can be obtained from a kiosk. Advantages for members include quick retrieval of bikes and unlimited 45-minute rentals. Short-term passes offer access to bikes for a 24-hour or 3-day time interval. Riders can avail and return the bikes at any of the 50 stations citywide.

Jason started this service in May 2014 and since then had been focusing on increasing the number of bikes as well as docking stations in order to increase convenience and accessibility for his customers. Despite this expansion, customer retention remained an issue. As Jason recalled, "We had planned to put in the investment for a year to lay out the infrastructure necessary for the customers to start using it. We had a strategy to make sure that the retention levels remain high to make this model self-sustainable. However, it worked otherwise (i.e., the customer base didn't catch up with the rate of the infrastructure expansion)."

A private service would have had three alternatives to curb this problem: get sponsors on board, increase service charges, or expand the pool of customers. Price hikes were not an option for Jason as this was a publicly sponsored initiative with the goal of providing affordable transportation to all. As for increasing the customer base, they had to decide upon a marketing channel that guarantees broad reach on low cost incurred.

Nancy, a marketer who had worked in the corporate sector for ten years, and Eric, a data analyst, were explicitly hired to find a way to make things work around this problem. The advantage on their side was that they were provided with the dataset of transaction history and thus they didn't had to go through the hassle of conducting marketing research to gather data.

Nancy realized that attracting recurring customers on a minimal budget required understanding the customers in the first place (i.e., persona). As she stated, "Understanding the persona of your brand is essential, as it helps you reach a targeted audience which is likely to convert at a higher probability. Moreover, this also helps in reaching out to sponsors who target a similar persona. This two-fold approach can make our bottom line positive."

As Nancy and Eric contemplated the problem at hand, they had questions like the following: Which attribute correlates the best with trip duration and number of trips? Which age generation adapts the most to our service?

Following is the data dictionary of the *Trips* dataset that was provided to Nancy and Eric:

Table 1-1. *Data Dictionary for the Trips Data from Cycles Share Dataset*

Feature name	Description
trip_id	Unique ID assigned to each trip
Starttime	Day and time when the trip started, in PST
Stoptime	Day and time when the trip ended, in PST
Bikeid	ID attached to each bike
Tripduration	Time of trip in seconds
from_station_name	Name of station where the trip originated
to_station_name	Name of station where the trip terminated
from_station_id	ID of station where trip originated
to_station_id	ID of station where trip terminated
Usertype	Value can include either of the following: short-term pass holder or member
Gender	Gender of the rider
Birthyear	Birth year of the rider

Exercises for this chapter required Eric to install the packages shown in Listing 1-1. He preferred to import all of them upfront to avoid bottlenecks while implementing the code snippets on your local machine.

However, for Eric to import these packages in his code, he needed to install them in the first place. He did so as follows:

1. Opened terminal/shell
2. Navigated to his code directory using terminal/shell
3. Installed `pip`:

   ```
   python get-pip.py
   ```

4. Installed each package separately, for example:

   ```
   pip install pandas
   ```

Listing 1-1. Importing Packages Required for This Chapter

```
%matplotlib inline

import random
import datetime
import pandas as pd
import matplotlib.pyplot as plt
import statistics
```

```
import numpy as np
import scipy
from scipy import stats
import seaborn
```

Performing Exploratory Data Analysis

Eric recalled to have explained Exploratory Data Analysis in the following words:

> *What do I mean by exploratory data analysis (EDA)? Well, by this I mean to see the data visually. Why do we need to see the data visually? Well, considering that you have 1 million observations in your dataset then it won't be easy for you to understand the data just by looking at it, so it would be better to plot it visually. But don't you think it's a waste of time? No not at all, because understanding the data lets us understand the importance of features and their limitations.*

Feature Exploration

Eric started off by loading the data into memory (see Listing 1-2).

Listing 1-2. Reading the Data into Memory

```
data = pd.read_csv('examples/trip.csv')
```

Nancy was curious to know how big the data was and what it looked like. Hence, Eric wrote the code in Listing 1-3 to print some initial observations of the dataset to get a feel of what it contains.

Listing 1-3. Printing Size of the Dataset and Printing First Few Rows

```
print len(data)
data.head()
```

Output

```
236065
```

***Table 1-2.** Print of Observations in the First Seven Columns of Dataset*

trip_id	starttime	stoptime	bikeid	tripduration	from_station_name	to_station_name
431	10/13/2014 10:31	10/13/2014 10:48	SEA00298	985.935	2nd Ave & Spring St	Occidental Park/ Occidental Ave S & S Washing...
432	10/13/2014 10:32	10/13/2014 10:48	SEA00195	926.375	2nd Ave & Spring St	Occidental Park/ Occidental Ave S & S Washing...
433	10/13/2014 10:33	10/13/2014 10:48	SEA00486	883.831	2nd Ave & Spring St	Occidental Park/ Occidental Ave S & S Washing...
434	10/13/2014 10:34	10/13/2014 10:48	SEA00333	865.937	2nd Ave & Spring St	Occidental Park/ Occidental Ave S & S Washing...
435	10/13/2014 10:34	10/13/2014 10:49	SEA00202	923.923	2nd Ave & Spring St	Occidental Park/ Occidental Ave S & S Washing...

***Table 1-3.** Print of Observations in the Last five Columns of Dataset*

from_station_id	to_station_id	usertype	gender	birthyear
CBD-06	PS-04	Member	Male	1960.0
CBD-06	PS-04	Member	Male	1970.0
CBD-06	PS-04	Member	Female	1988.0
CBD-06	PS-04	Member	Female	1977.0
CBD-06	PS-04	Member	Male	1971.0

After looking at Table 1-2 and Table 1-3 Nancy noticed that `tripduration` is represented in seconds. Moreover, the unique identifiers for `bike`, `from_station`, and `to_station` are in the form of strings, contrary to those for `trip identifier` which are in the form of integers.

Types of variables

Nancy decided to go an extra mile and allocated data type to each feature in the dataset.

Table 1-4. *Nancy's Approach to Classifying Variables into Data Types*

Feature name	Variable type

`trip_id`
`bikeid`
`tripduration`
`from_station_id`
`to_station_id`
`birthyear`
	Numbers

`Starttime` `Stoptime`	Date

`from_station_name to_station_name`
`Usertype`
`Gender`
	Text

After looking at the feature classification in Table 1-4 Eric noticed that Nancy had correctly identified the data types and thus it seemed to be an easy job for him to explain what variable types mean. As Eric recalled to have explained the following:

> *In normal everyday interaction with data we usually represent numbers as integers, text as strings, True/False as Boolean, etc. These are what we refer to as data types. But the lingo in machine learning is a bit more granular, as it splits the data types we knew earlier into variable types. Understanding these variable types is crucial in deciding upon the type of charts while doing exploratory data analysis or while deciding upon a suitable machine learning algorithm to be applied on our data.*

Continuous/Quantitative Variables

A continuous variable can have an infinite number of values within a given range. Unlike discrete variables, they are not countable. Before exploring the types of continuous variables, let's understand what is meant by a true zero point.

True Zero Point

If a level of measurement has a true zero point, then a value of 0 means you have nothing. Take, for example, a ratio variable which represents the number of cupcakes bought. A value of 0 will signify that you didn't buy even a single cupcake. The true zero point is a strong discriminator between interval and ratio variables.

Let's now explore the different types of continuous variables.

Interval Variables

Interval variables exist around data which is continuous in nature and has a numerical value. Take, for example, the temperature of a neighborhood measured on a daily basis. Difference between intervals remains constant, such that the difference between 70 Celsius and 50 Celsius is the same as the difference between 80 Celsius and 100 Celsius. We can compute the mean and median of interval variables however they don't have a true zero point.

Ratio Variables

Properties of interval variables are very similar to those of ratio variables with the difference that in ratio variables a 0 indicates the absence of that measurement. Take, for example, distance covered by cars from a certain neighborhood. Temperature in Celsius is an interval variable, so having a value of 0 Celsius does not mean absence of temperature. However, notice that a value of 0 KM will depict no distance covered by the car and thus is considered as a ratio variable. Moreover, as evident from the name, ratios of measurements can be used as well such that a distance covered of 50 KM is twice the distance of 25 KM covered by a car.

Discrete Variables

A discrete variable will have finite set of values within a given range. Unlike continuous variables those are countable. Let's look at some examples of discrete variables which are categorical in nature.

Ordinal Variables

Ordinal variables have values that are in an order from lowest to highest or vice versa. These levels within ordinal variables can have unequal spacing between them. Take, for example, the following levels:

1. Primary school
2. High school
3. College
4. University

The difference between primary school and high school in years is definitely not equal to the difference between high school and college. If these differences were constant, then this variable would have also qualified as an interval variable.

Nominal Variables

Nominal variables are categorical variables with no intrinsic order; however, constant differences between the levels exist. Examples of nominal variables can be gender, month of the year, cars released by a manufacturer, and so on. In the case of month of year, each month is a different level.

Dichotomous Variables

Dichotomous variables are nominal variables which have only two categories or levels. Examples include

- Age: under 24 years, above 24 years
- Gender: male, female

Lurking Variable

A lurking variable is not among exploratory (i.e., independent) or response (i.e., dependent) variables and yet may influence the interpretations of relationship among these variables. For example, if we want to predict whether or not an applicant will get admission in a college on the basis of his/her gender. A possible lurking variable in this case can be the name of the department the applicant is seeking admission to.

Demographic Variable

Demography (from the Greek word meaning "description of people") is the study of human populations. The discipline examines size and composition of populations as well as the movement of people from locale to locale. Demographers also analyze the effects of population growth and its control. A demographic variable is a variable that is collected by researchers to describe the nature and distribution of the sample used with inferential statistics. Within applied statistics and research, these are variables such as age, gender, ethnicity, socioeconomic measures, and group membership.

Dependent and Independent Variables

An independent variable is also referred to as an exploratory variable because it is being used to explain or predict the dependent variable, also referred to as a response variable or outcome variable.

Taking the dataset into consideration, what are the dependent and independent variables? Let's say that Cycle Share System's management approaches you and asks you to build a system for them to predict the trip duration beforehand so that the supply

of cycles can be ensured. In that case, what is your dependent variable? Definitely tripduration. And what are the independent variables? Well, these variables will comprise of the features which we believe influence the dependent variable (e.g., usertype, gender, and time and date of the day).

Eric asked Nancy to classify the features in the variable types he had just explained.

Table 1-5. *Nancy's Approach to Classifying Variables into Variable Types*

Feature name	Variable type

`trip_id`
`bikeid`
`tripduration`
`from_station_id`
`to_station_id`
`birthyear`
	Continuous

`Starttime` `Stoptime`	DateTime
`from_station_name` `to_station_name`	String
`Usertype gender`	Nominal

Nancy now had a clear idea of the variable types within machine learning, and also which of the features qualify for which of those variable types (see Table 1-5). However despite of looking at the initial observations of each of these features (see Table 1-2) she couldn't deduce the depth and breadth of information that each of those tables contains. She mentioned this to Eric, and Eric, being a data analytics guru, had an answer: perform univariate analysis on features within the dataset.

Univariate Analysis

Univariate comes from the word "uni" meaning one. This is the analysis performed on a single variable and thus does not account for any sort of relationship among exploratory variables.

Eric decided to perform univariate analysis on the dataset to better understand the features in isolation (see Listing 1-4).

Listing 1-4. Determining the Time Range of the Dataset

```
data = data.sort_values(by='starttime')
data.reset_index()
print 'Date range of dataset: %s - %s'%(data.ix[1, 'starttime'],
data.ix[len(data)-1, 'stoptime'])
```

Output

```
Date range of dataset: 10/13/2014 10:32 - 9/1/2016 0:20
```

Eric knew that Nancy would have a hard time understanding the code so he decided to explain the ones that he felt were complex in nature. In regard to the code in Listing 1-4, Eric explained the following:

> *We started off by sorting the data frame by* `starttime`*. Do note that data frame is a data structure in Python in which we initially loaded the data in Listing 1-2. Data frame helps arrange the data in a tabular form and enables quick searching by means of hash values. Moreover, data frame comes up with handy functions that make lives easier when doing analysis on data. So what sorting did was to change the position of records within the data frame, and hence the change in positions disturbed the arrangement of the indexes which were earlier in an ascending order. Hence, considering this, we decided to reset the indexes so that the ordered data frame now has indexes in an ascending order. Finally, we printed the date range that started from the first value of* `starttime` *and ended with the last value of* `stoptime`*.*

Eric's analysis presented two insights. One is that the data ranges from October 2014 up till September 2016 (i.e., three years of data). Moreover, it seems like the cycle sharing service is usually operational beyond the standard 9 to 5 business hours.

Nancy believed that short-term pass holders would avail more trips than their counterparts. She believed that most people would use the service on a daily basis rather than purchasing the long term membership. Eric thought otherwise; he believed that new users would be short-term pass holders however once they try out the service and become satisfied would ultimately avail the membership to receive the perks and benefits offered. He also believed that people tend to give more weight to services they have paid for, and they make sure to get the maximum out of each buck spent. Thus, Eric decided to plot a bar graph of trip frequencies by user type to validate his viewpoint (see Listing 1-5). But before doing so he made a brief document of the commonly used charts and situations for which they are a best fit to (see Appendix A for a copy). This chart gave Nancy his perspective for choosing a bar graph for the current situation.

Listing 1-5. Plotting the Distribution of User Types

```
groupby_user = data.groupby('usertype').size()
groupby_user.plot.bar(title = 'Distribution of user types')
```

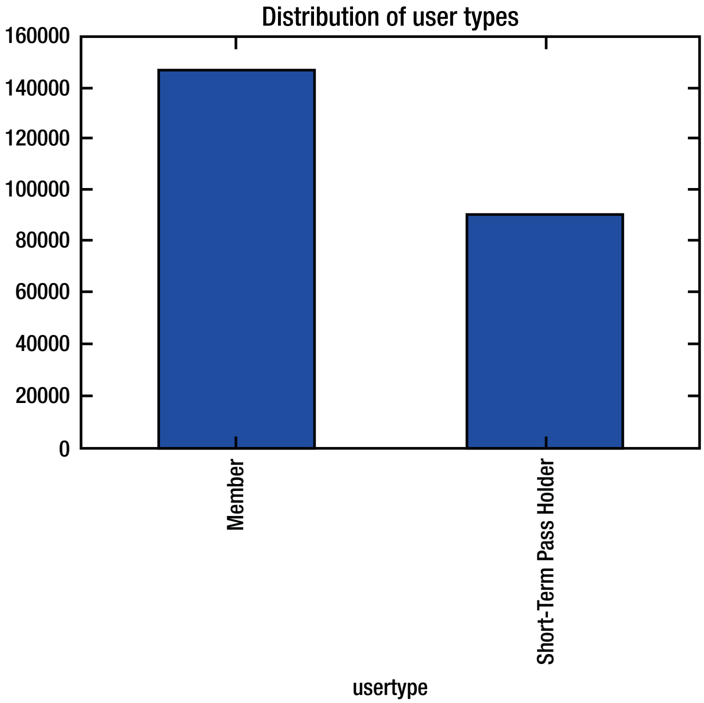


Figure 1-1. *Bar graph signifying the distribution of user types*

Nancy didn't understand the code snippet in Listing 1-5. She was confused by the functionality of groupby and size methods. She recalled asking Eric the following: "I can understand that groupby groups the data by a given field, that is, usertype, in the current situation. But what do we mean by size? Is it the same as count, that is, counts trips falling within each of the grouped usertypes?"

Eric was surprised by Nancy's deductions and he deemed them to be correct. However, the bar graph presented insights (see Figure 1-1) in favor of Eric's view as the members tend to avail more trips than their counterparts.

Nancy had recently read an article that talked about the gender gap among people who prefer riding bicycles. The article mentioned a cycle sharing scheme in UK where 77% of the people who availed the service were men. She wasn't sure if similar phenomenon exists for people using the service in United States. Hence Eric came up with the code snippet in Listing 1-6 to answer the question at hand.

Listing 1-6. Plotting the Distribution of Gender

```
groupby_gender = data.groupby('gender').size()
groupby_gender.plot.bar(title = 'Distribution of genders')
```

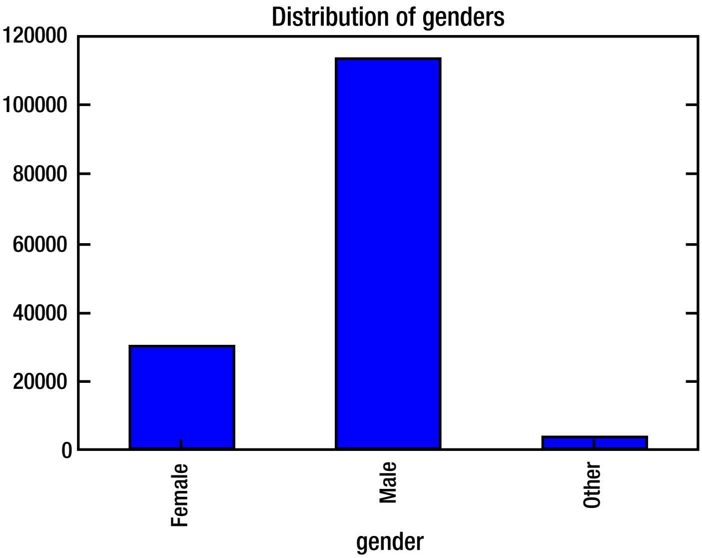


Figure 1-2. *Bar graph signifying the distribution of genders*

Figure 1-2 revealed that the gender gap resonates in states as well. Males seem to dominate the trips taken as part of the program.

Nancy, being a marketing guru, was content with the analysis done so far. However she wanted to know more about her target customers to whom to company's marketing message will be targetted to. Thus Eric decided to come up with the distribution of birth years by writing the code in Listing 1-7. He believed this would help the Nancy understand the age groups that are most likely to ride a cycle or the ones that are more prone to avail the service.

Listing 1-7. Plotting the Distribution of Birth Years

```
data = data.sort_values(by='birthyear')
groupby_birthyear = data.groupby('birthyear').size()
groupby_birthyear.plot.bar(title = 'Distribution of birth years',
figsize = (15,4))
```

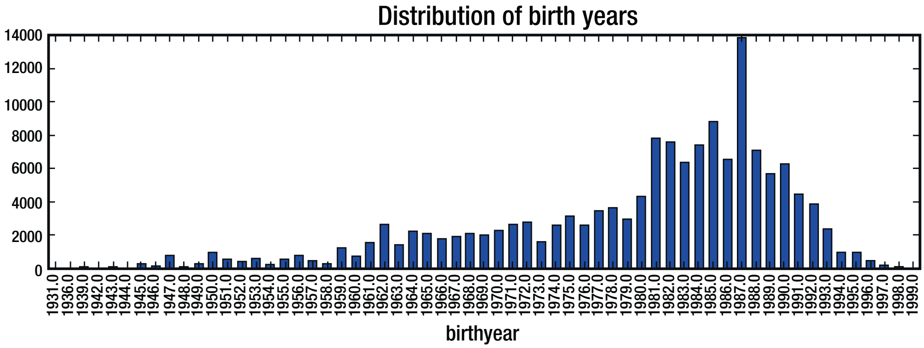


Figure 1-3. *Bar graph signifying the distribution of birth years*

Figure 1-3 provided a very interesting illustration. Majority of the people who had subscribed to this program belong to Generation Y (i.e., born in the early 1980s to mid to late 1990s, also known as millennials). Nancy had recently read the reports published by *Elite Daily* and *CrowdTwist* which said that millennials are the most loyal generation to their favorite brands. One reason for this is their willingness to share thoughts and opinions on products/services. These opinions thus form a huge corpus of experiences—enough information for the millenials to make a conscious decision, a decision they will remain loyal to for a long period. Hence Nancy was convinced that most millennials would be members rather than short-term pass holders. Eric decided to populate a bar graph to see if Nancy's deduction holds true.

Listing 1-8. Plotting the Frequency of Member Types for Millenials

```
data_mil = data[(data['birthyear'] >= 1977) & (data['birthyear']<=1994)]
groupby_mil = data_mil.groupby('usertype').size()
groupby_mil.plot.bar(title = 'Distribution of user types')
```

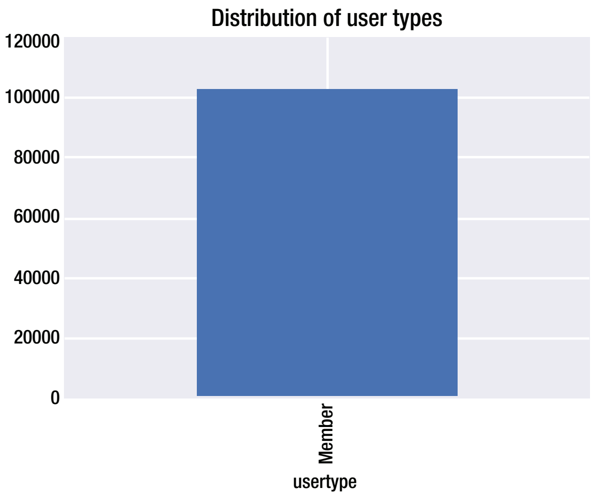


Figure 1-4. *Bar graph of member types for millenials*

After looking at Figure 1-4 Eric was surprised to see that Nancy's deduction appeared to be valid, and Nancy made a note to make sure that the brand engaged millennials as part of the marketing plan.

Eric knew that more insights can pop up when more than one feature is used as part of the analysis. Hence, he decided to give Nancy a sneak peek at multivariate analysis before moving forward with more insights.

Multivariate Analysis

Multivariate analysis refers to incorporation of multiple exploratory variables to understand the behavior of a response variable. This seems to be the most feasible and realistic approach considering the fact that entities within this world are usually interconnected. Thus the variability in response variable might be affected by the variability in the interconnected exploratory variables.

Nancy believed males would dominate females in terms of the trips completed. The graph in Figure 1-2, which showed that males had completed far more trips than any other gender types, made her embrace this viewpoint. Eric thought that the best approach to validate this viewpoint was a stacked bar graph (i.e., a bar graph for birth year, but each bar having two colors, one for each gender) (see Figure 1-5).

Listing 1-9. Plotting the Distribution of Birth Years by Gender Type

```
groupby_birthyear_gender = data.groupby(['birthyear', 'gender'])
['birthyear'].count().unstack('gender').fillna(0)
groupby_birthyear_gender[['Male','Female','Other']].plot.bar(title =
'Distribution of birth years by Gender', stacked=True, figsize = (15,4))
```

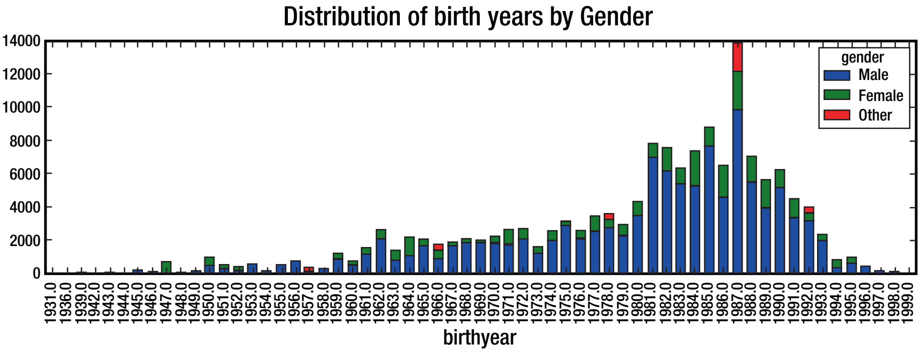


Figure 1-5. *Bar graph signifying the distribution of birth years by gender type*

The code snippet in Listing 1-9 brought up some new aspects not previously highlighted.

> *We at first transformed the data frame by unstacking, that is, splitting, the gender column into three columns, that is, Male, Female, and Other. This meant that for each of the birth years we had the trip count for all three gender types. Finally, a stacked bar graph was created by using this transformed data frame.*

It seemed as if males were dominating the distribution. It made sense as well. No? Well, it did; as seen earlier, that majority of the trips were availed by males, hence this skewed the distribution in favor of males. However, subscribers born in 1947 were all females. Moreover, those born in 1964 and 1994 were dominated by females as well. Thus Nancy's hypothesis and reasoning did hold true.

The analysis in Listing 1-4 had revealed that all millennials are members. Nancy was curious to see what the distribution of user type was for the other age generations. Is it that the majority of people in the other age generations were short-term pass holders? Hence Eric brought a stacked bar graph into the application yet again (see Figure 1-6).

Listing 1-10. Plotting the Distribution of Birth Years by User Types

```
groupby_birthyear_user = data.groupby(['birthyear', 'usertype'])
['birthyear'].count().unstack('usertype').fillna(0)

groupby_birthyear_user['Member'].plot.bar(title = 'Distribution of birth
years by Usertype', stacked=True, figsize = (15,4))
```

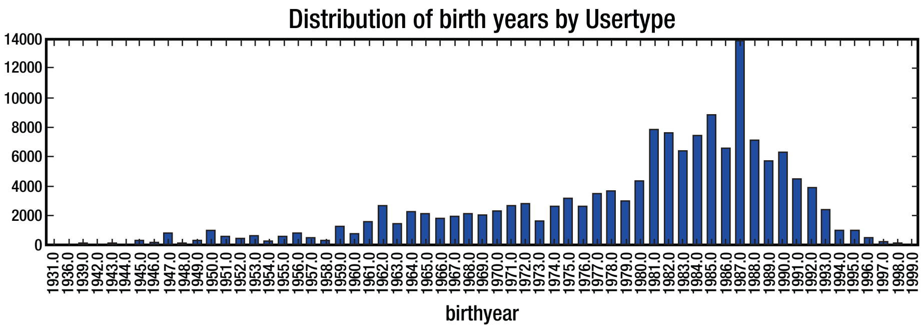


Figure 1-6. *Bar graph signifying the distribution of birth years by user types*

Whoa! Nancy was surprised to see the distribution of only one user type and not two (i.e., membership and short-term pass holders)? Does this mean that birth year information was only present for only one user type? Eric decided to dig in further and validate this (see Listing 1-11).

Listing 1-11. Validation If We Don't Have Birth Year Available for Short-Term Pass Holders

```
data[data['usertype']=='Short-Term Pass Holder']['birthyear'].isnull().
values.all()
```

Output

```
True
```

In the code in Listing 1-11, Eric first sliced the data frame to consider only short-term pass holders. Then he went forward to find out if all the values in birth year are missing (i.e., null) for this slice. Since that is the case, Nancy's initially inferred hypothesis was true—that birth year data is only available for members. This made her recall her prior deduction about the brand loyalty of millennials. Hence the output for Listing 1-11 nullifies Nancy's deduction made after the analysis in Figure 1-4. This made Nancy sad, as the loyalty of millenials can't be validated from the data at hand. Eric believed that members have to provide details like birth year when applying for the membership, something which is not a prerequisite for short-term pass holders. Eric decided to test his deduction by checking if gender is available for short-term pass holders or not for which he wrote the code in Listing 1-12.

Listing 1-12. Validation If We Don't Have Gender Available for Short-Term Pass Holders

```
data[data['usertype']=='Short-Term Pass Holder']['gender'].isnull().values.
all()
```

Output

```
True
```

Thus Eric concluded that we don't have the demographic variables for user type 'Short-Term Pass holders.'

Nancy was interested to see as to how the frequency of trips vary across date and time (i.e., a time series analysis). Eric was aware that trip start time is given with the data, but for him to make a time series plot, he had to transform the date from string to date time format (see Listing 1-13). He also decided to do more: that is, split the datetime into date components (i.e., year, month, day, and hour).

Listing 1-13. Converting String to datetime, and Deriving New Features

```
List_ = list(data['starttime'])

List_ = [datetime.datetime.strptime(x, "%m/%d/%Y %H:%M") for x in List_]
data['starttime_mod'] = pd.Series(List_,index=data.index)
data['starttime_date'] = pd.Series([x.date() for x in List_],index=data.index)
data['starttime_year'] = pd.Series([x.year for x in List_],index=data.index)
data['starttime_month'] = pd.Series([x.month for x in List_],index=data.index)
data['starttime_day'] = pd.Series([x.day for x in List_],index=data.index)
data['starttime_hour'] = pd.Series([x.hour for x in List_],index=data.index)
```

Eric made sure to explain the piece of code in Listing 1-13 as he had explained to Nancy:

> *At first we converted start time column of the dataframe into a list. Next we converted the string dates into python datetime objects. We then converted the list into a series object and converted the dates from datetime object to pandas date object. The time components of year, month, day and hour were derived from the list with the datetime objects.*

Now it was time for the time series analysis of the frequency of trips over all days provided within the dataset (see Listing 1-14).

Listing 1-14. Plotting the Distribution of Trip Duration over Daily Time

```
data.groupby('starttime_date')['tripduration'].mean().plot.bar(title =
'Distribution of Trip duration by date', figsize = (15,4))
```

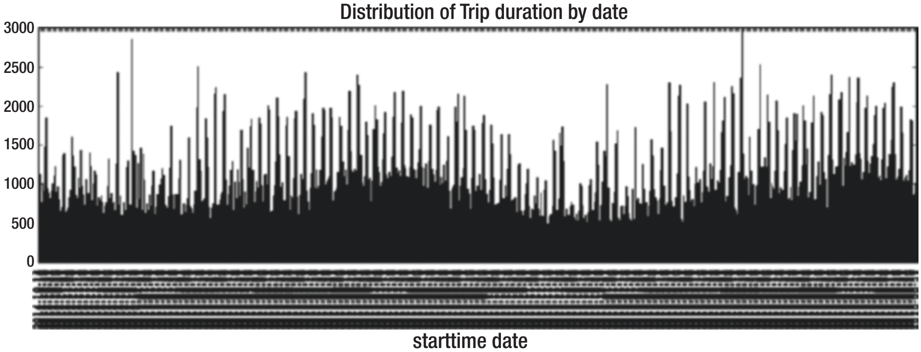


Figure 1-7. *Bar graph signifying the distribution of trip duration over daily time*

Wow! There seems to be a definitive pattern of trip duration over time.

Time Series Components

Eric decided to brief Nancy about the types of patterns that exist in a time series analysis. This he believed would help Nancy understand the definite pattern in Figure 1-7.

Seasonal Pattern

A seasonal pattern (see Figure 1-8) refers to a seasonality effect that incurs after a fixed known period. This period can be week of the month, week of the year, month of the year, quarter of the year, and so on. This is the reason why seasonal time series are also referred to as periodic time series.

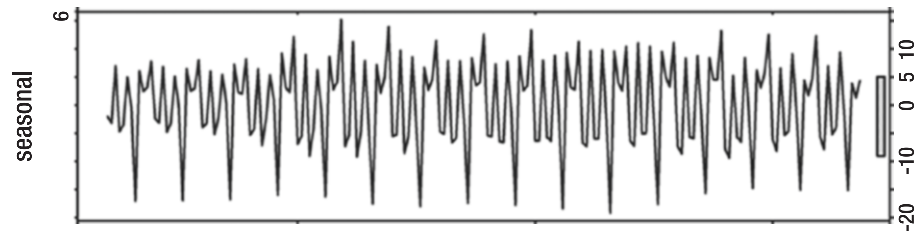


Figure 1-8. *Illustration of seasonal pattern*

Cyclic Pattern

A cyclic pattern (see Figure 1-9) is different from a seasonal pattern in the notion that the patterns repeat over non-periodic time cycles.

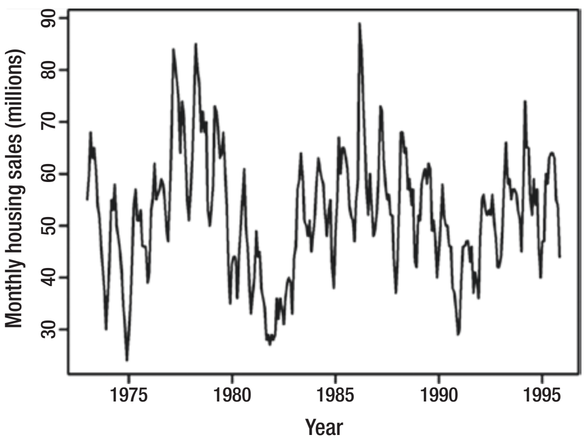


Figure 1-9. *Illustration of cyclic pattern*

Trend

A trend (see Figure 1-10) is a long-term increase or decrease in a continuous variable. This pattern might not be exactly linear over time, but when smoothing is applied it can generalize into either of the directions.

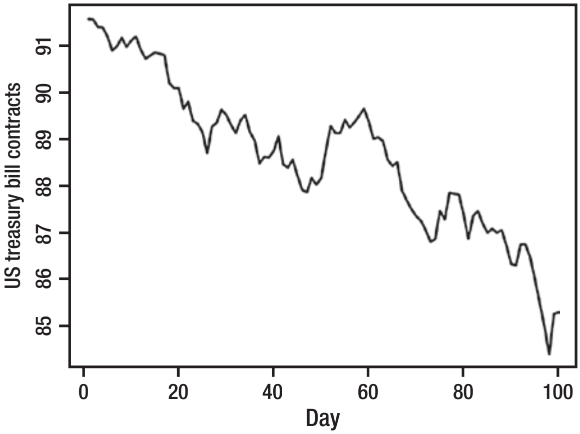


Figure 1-10. *Illustration of trend*

Eric decided to test Nancy's concepts on time series, so he asked her to provide her thoughts on the time series plot in Figure 1-7. "What do you think of the time series plot? Is the pattern seasonal or cyclic? Seasonal is it right?"

Nancy's reply amazed Eric once again. She said the following:

> *Yes it is because the pattern is repeating over a fixed interval of time—that is, seasonality. In fact, we can split the distribution into three distributions. One pattern is the seasonality that is repeating over time. The second one is a flat density distribution. Finally, the last pattern is the lines (that is, the hikes) over that density function. In case of time series prediction we can make estimations for a future time using both of these distributions and add up in order to predict upon a calculated confidence interval.*

On the basis of her deduction it seemed like Nancy's grades in her statistics elective course had paid off. Nancy wanted answers to many more of her questions. Hence she decided to challenge the readers with the Exercises that follow.

EXERCISES

1. Determine the distribution of number of trips by year. Do you see a specific pattern?
2. Determine the distribution of number of trips by month. Do you see a specific pattern?
3. Determine the distribution of number of trips by day. Do you see a specific pattern?
4. Determine the distribution of number of trips by day. Do you see a specific pattern?
5. Plot a frequency distribution of trips on a daily basis.

Measuring Center of Measure

Eric believed that measures like mean, median, and mode help give a summary view of the features in question. Taking this into consideration, he decided to walk Nancy through the concepts of center of measure.

Mean

Mean in layman terms refers to the averaging out of numbers Mean is highly affected by outliers, as the skewness introduced by outliers will pull the mean toward extreme values.

- Symbol:
  - μ-> Parameter -> population mean
  - $\acute{x}$ -> Statistic -> sample mean
- Rules of mean:
  - $\mu_{a+bx} = a + b\mu_x$
  - $\mu_{x+y} = \mu_x + \mu_y$

We will be using **statistics.mean(data)** in our coding examples. This will return the sample arithmetic mean of data, a sequence or iterator of real-valued numbers.

Mean exists in two major variants.

Arithmetic Mean

An arithmetic mean is simpler than a geometric mean as it averages out the numbers (i.e., it adds all the numbers and then divides the sum by the frequency of those numbers). Take, for example, the grades of ten students who appeared in a mathematics test.

78, 65, 89, 93, 87, 56, 45, 73, 51, 81

Calculating the arithmetic mean will mean

$$mean = \frac{78+65+89+93+87+56+45+73+51+81}{10} = 71.8$$

Hence the arithmetic mean of scores taken by students in their mathematics test was 71.8. Arithmetic mean is most suitable in situations when the observations (i.e., math scores) are independent of each other. In this case it means that the score of one student in the test won't affect the score that another student will have in the same test.

Geometric Mean

As we saw earlier, arithmetic mean is calculated for observations which are independent of each other. However, this doesn't hold true in the case of a geometric mean as it is used to calculate mean for observations that are dependent on each other. For example, suppose you invested your savings in stocks for five years. Returns of each year will be invested back in the stocks for the subsequent year. Consider that we had the following returns in each one of the five years:

60%, 80%, 50%, -30%, 10%

Are these returns dependent on each other? Well, yes! Why? Because the investment of the next year is done on the capital garnered from the previous year, such that a loss in the first year will mean less capital to invest in the next year and vice versa. So, yes, we will be calculating the geometric mean. But how? We will do so as follows:

$$[(0.6 + 1) * (0.8 + 1) * (0.5 + 1) * (-0.3 + 1) * (0.1 + 1)]^{1/5} - 1 = 0.2713$$

Hence, an investment with these returns will yield a return of 27.13% by the end of the fifth year. Looking at the calculation above, you can see that at first we first converted percentages into decimals. Next we added 1 to each of them to nullify the effects brought on by the negative terms. Then we multiplied all terms among themselves and applied a power to the resultant. The power applied was 1 divided by the frequency of observations (i.e., five in this case). In the end we subtracted the result by 1. Subtraction was done to nullify the effect introduced by an addition of 1, which we did initially with each term. The subtraction by 1 would not have been done had we not added 1 to each of the terms (i.e., yearly returns).

Median

Median is a measure of central location alongside mean and mode, and it is less affected by the presence of outliers in your data. When the frequency of observations in the data is odd, the middle data point is returned as the median.

In this chaapter we will use **statistics.median(data)** to calculate the median. This returns the median (middle value) of numeric data if frequency of values is odd and otherwise mean of the middle values if frequency of values is even using “mean of middle two” method. If data is empty, StatisticsError is raised.

Mode

Mode is suitable on data which is discrete or nominal in nature. Mode returns the observation in the dataset with the highest frequency. Mode remains unaffected by the presence of outliers in data.

Variance

Variance represents variability of data points about the mean. A high variance means that the data is highly spread out with a small variance signifying the data to be closely clustered.

1. Symbol: σ_x^2
2. Formula:

   a. $$\frac{\sum(X-X')^2}{n-1}$$

   b. $$\sigma_x^2 = \sum(x_i - \mu_x)^2 p_i$$

3. Why n-1 beneath variance calculation? *The sample variance averages out to be smaller than the population variance; hence, degrees of freedom is accounted for as the conversion factor.*
4. Rules of variance:

   i. $\sigma_{a+bx}^2 = b^2\sigma_x^2$

   ii. $\sigma_{x+y}^2 = \sigma_x^2 + \sigma_y^2$ (If X and Y are independent variables)

   $$\sigma_{x-y}^2 = \sigma_x^2 + \sigma_y^2$$

   iii. $\sigma_{x+y}^2 = \sigma_x^2 + \sigma_y^2 + 2r\sigma_x\sigma_y$ (if X and Y have correlation r)

   $$\sigma_{x+y}^2 = \sigma_x^2 + \sigma_y^2 + 2r\sigma_x\sigma_y$$

We will be incorporating **statistics.variance(data, xbar=None)** to calculate variance in our coding exercises. This will return the sample variance across at least two real-valued numbered series.

Standard Deviation

Standard deviation, just like variance, also captures the spread of data along the mean. The only difference is that it is a square root of the variance. This enables it to have the same unit as that of the data and thus provides convenience in inferring explanations from insights. Standard deviation is highly affected by outliers and skewed distributions.

- Symbol: $\boldsymbol{\sigma}$
- Formula: $\sqrt{\sigma^2}$

We measure standard deviation instead of variance because

- *It is the natural measure of spread in a Normal distribution*
- *Same units as original observations*

Changes in Measure of Center Statistics due to Presence of Constants

Let's evaluate how measure of center statistics behave when data is transformed by the introduction of constants. We will evaluate the outcomes for mean, median, IQR (interquartile range), standard deviation, and variance. Let's first start with what behavior each of these exhibits when a constant "a" is added or subtracted from each of these.

Addition: *Adding a*

- $\acute{x}_{new} = a + \acute{x}$
- $median_{new} = a + median$
- $IQR_{new} = a + IQR$
- $s_{new} = s$
- $\sigma^2_{x\,new} = \sigma^2_x$

Adding a constant to each of the observations affected the mean, median, and IQR. However, standard deviation and variance remained unaffected. Note that the same behavior will come through when observations within the data are subtracted from a constant. Let's see if the same behavior will repeat when we multiply a constant (i.e., "b") to each observation within the data.

Multiplication: *Multiplying b*

- $x'_{new} = bx'$
- $median_{new} = bmedian$
- $IQR_{new} = bIQR$
- $s_{new} = bs$
- $\sigma^2_{x\,new} = b^2\sigma^2_x$

Wow! Multiplying a constant to each observation within the data changed all five measures of center statistics. Do note that you will achieve the same effect when all observations within the data are divided by a constant term.

After going through the description of center of measures, Nancy was interested in understanding the trip durations in detail. Hence Eric came up with the idea to calculate the mean and median trip durations. Moreover, Nancy wanted to determine the station from which most trips originated in order to run promotional campaigns for existing customers. Hence Eric decided to determine the mode of 'from_station_name' field.

■ **Note** Determining the measures of centers using the statistics package will require us to transform the input data structure to a list type.

Listing 1-15. Determining the Measures of Center Using Statistics Package

```
trip_duration = list(data['tripduration'])
station_from = list(data['from_station_name'])
print 'Mean of trip duration: %f'%statistics.mean(trip_duration)
print 'Median of trip duration: %f'%statistics.median(trip_duration)
print 'Mode of station originating from: %s'%statistics.mode(station_from)
```

Output

```
Mean of trip duration: 1202.612210
Median of trip duration: 633.235000
Mode of station originating from: Pier 69 / Alaskan Way & Clay St
```

The output of Listing 1-15 revealed that most trips originated from Pier 69/Alaskan Way & Clay St station. Hence this was the ideal location for running promotional campaigns targeted to existing customers. Moreover, the output showed the mean to be greater than that of the mean. Nancy was curious as to why the average (i.e., mean) is greater than the central value (i.e., median). On the basis of what she had read, she realized that this might be either due to some extreme values after the median or due to the majority of values lying after the median. Eric decided to plot a distribution of the trip durations (see Listing 1-16) in order to determine which premise holds true.

Listing 1-16. Plotting Histogram of Trip Duration

```
data['tripduration'].plot.hist(bins=100, title='Frequency distribution of
Trip duration')
plt.show()
```

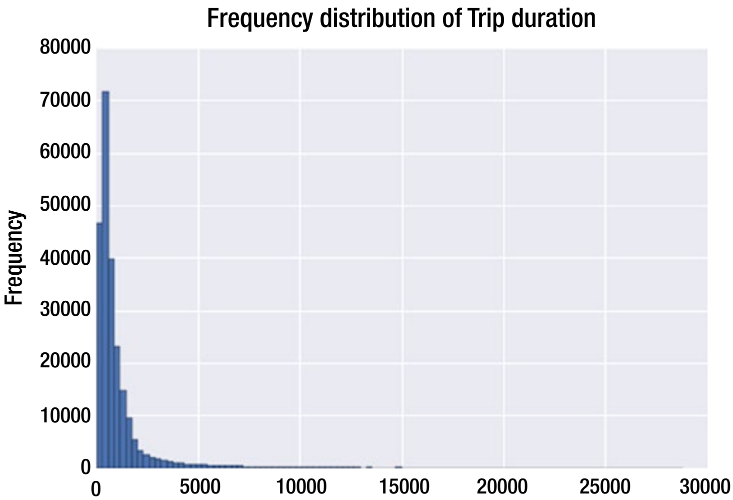


Figure 1-11. *Frequency distribution of trip duration*

The distribution in Figure 1-11 has only one peak (i.e., mode). The distribution is not symmetric and has majority of values toward the right-hand side of the mode. These extreme values toward the right are negligible in quantity, but their extreme nature tends to pull the mean toward themselves. Thus the reason why the mean is greater than the median.

The distribution in Figure 1-11 is referred to as a normal distribution.

The Normal Distribution

Normal distribution, or in other words Gaussian distribution, is a continuous probability distribution that is bell shaped. The important characteristic of this distribution is that the mean lies at the center of this distribution with a spread (i.e., standard deviation) around it. The majority of the observations in normal distribution lie around the mean and fade off as they distance away from the mean. Some 68% of the observations lie within 1 standard deviation from the mean; 95% of the observations lie within 2 standard deviations from the mean, whereas 99.7% of the observations lie within 3 standard deviations from the mean. A normal distribution with a mean of zero and a standard deviation of 1 is referred to as a standard normal distribution. Figure 1-12 shows normal distribution along with confidence intervals.

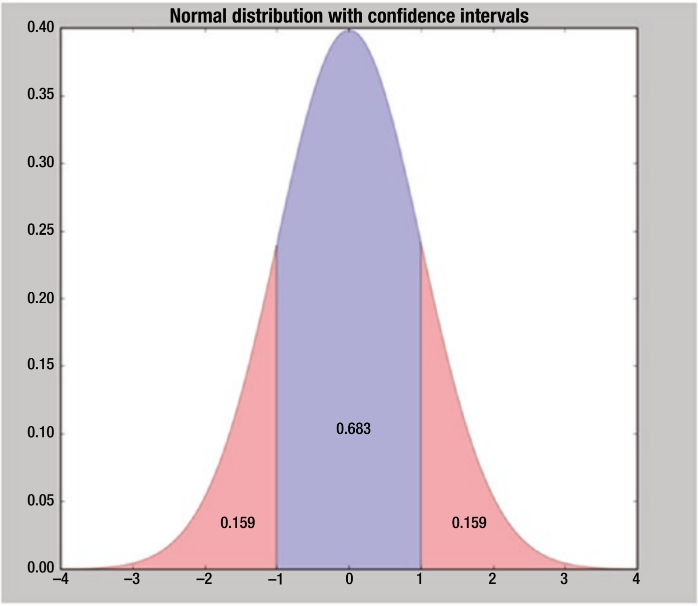


Figure 1-12. *Normal distribution and confidence levels*

These are the most common confidence levels:

Confidence level	Formula
68%	Mean ± 1 std.
95%	Mean ± 2 std.
99.7%	Mean ± 3 std.

Skewness

Skewness is a measure of the lack of symmetry. The normal distribution shown previously is symmetric and thus has no element of skewness. Two types of skewness exist (i.e., positive and negative skewness).

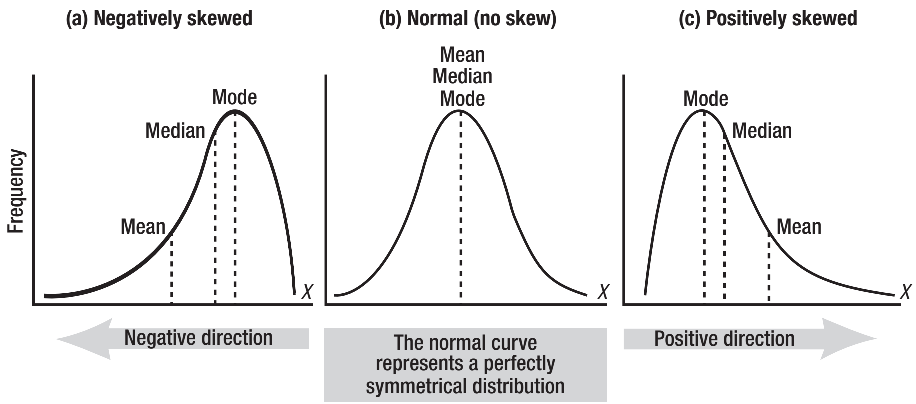


Figure 1-13. *Skewed and symmetric normal distributions*

As seen from Figure 1-13, a relationship exists among measure of centers for each one of the following variations:

- Symmetric distributions: *Mean* = *Median* = *Mode*
- Positively skewed: *Mean* < *Median* < *Mode*
- Negatively skewed: *Mean* > *Median* > *Mode*

Going through Figure 1-12 you will realize that the distribution in Figure 1-13(c) has a long tail on its right. This might be due to the presence of outliers.

Outliers

Outliers refer to the values distinct from majority of the observations. These occur either naturally, due to equipment failure, or because of entry mistakes.

In order to understand what outliers are, we need to look at Figure 1-14.

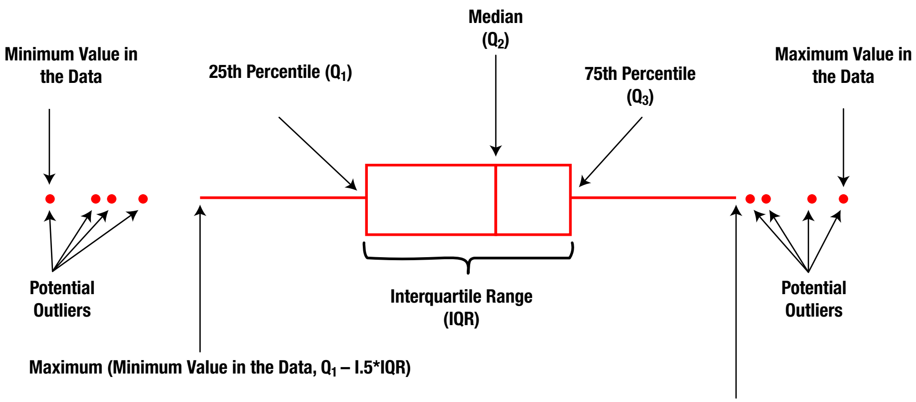


Figure 1-14. *Illustration of outliers using a box plot*

From Figure 1-14 we can see that the observations lying outside the whiskers are referred to as the outliers.

Listing 1-17. Interval of Values Not Considered Outliers

```
[Q1 - 1.5 (IQR) ,  Q3 + 1.5 (IQR) ] (i.e. IQR = Q3 - Q1)
```

Values not lying within this interval are considered outliers. Knowing the values of Q1 and Q3 is fundamental for this calculation to take place.

Is the presence of outliers good in the dataset? Usually not! So, how are we going to treat the outliers in our dataset? Following are the most common methods for doing so:

- *Remove the outliers:* This is only possible when the proportion of outliers to meaningful values is quite low, and the data values are not on a time series scale. If the proportion of outliers is high, then removing these values will hurt the richness of data, and models applied won't be able to capture the true essence that lies within. However, in case the data is of a time series nature, removing outliers from the data won't be feasible, the reason being that for a time series model to train effectively, data should be continuous with respect to time. Removing outliers in this case will introduce breaks within the continuous distribution.

- *Replace outliers with means:* Another way to approach this is by taking the mean of values lying with the interval shown in Figure 1-14, calculate the mean, and use these to replace the outliers. This will successfully transform the outliers in line with the valid observations; however, this will remove the anomalies that were otherwise present in the dataset, and their findings could present interesting insights.

- *Transform the outlier values:* Another way to cop up with outliers is to limit them to the upper and lower boundaries of acceptable data. The upper boundary can be calculated by plugging in the values of Q3 and IQR into Q3 + 1.5IQR and the lower boundary can be calculated by plugging in the values of Q1 and IQR into Q1 – 1.5IQR.

- *Variable transformation*: Transformations are used to convert the inherited distribution into a normal distribution. Outliers bring non-normality to the data and thus transforming the variable can reduce the influence of outliers. Methodologies of transformation include, but are not limited to, natural log, conversion of data into ratio variables, and so on.

Nancy was curious to find out whether outliers exist within our dataset—more precisely in the `tripduration` feature. For that Eric decided to first create a box plot (see Figure 1-15) by writing code in Listing 1-18 to see the outliers visually and then checked the same by applying the interval calculation method in Listing 1-19.

Listing 1-18. Plotting a Box plot of Trip Duration

```
box = data.boxplot(column=['tripduration'])
plt.show()
```

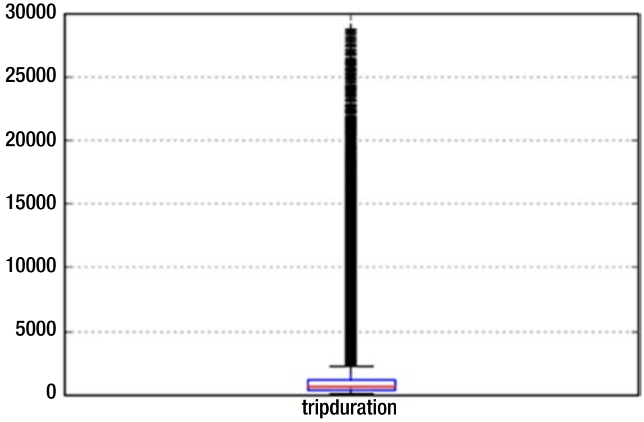


Figure 1-15. *Box plot of trip duration*

Nancy was surprised to see a huge number of outliers in trip duration from the box plot in Figure 1-15. She asked Eric if he could determine the proportion of trip duration values which are outliers. She wanted to know if outliers are a tiny or majority portion of the dataset. For that Eric wrote the code in Listing 1-19.

Listing 1-19. Determining Ratio of Values in Observations of tripduration Which Are Outliers

```
q75, q25 = np.percentile(trip_duration, [75 ,25])
iqr = q75 - q25
print 'Proportion of values as outlier: %f percent'%(
(len(data) - len([x for x in trip_duration if q75+(1.5*iqr)
>=x>= q25-(1.5*iqr)]))*100/float(len(data)))
```

Output

```
Proportion of values as outlier: 9.548218 percent
```

Eric explained the code in Listing 1-19 to Nancy as follows:

> *As seen in Figure 1-14, Q3 refers to the 75th percentile and Q1 refers to the 25th percentile. Hence we use the `numpy.percentile()` method to determine the values for Q1 and Q3. Next we compute the IQR by subtracting both of them. Then we determine the subset of values by applying the interval as specified in Listing 1-18. We then used the formula to get the number of outliers.*

Listing 1-20. Formula for Calculating Number of Outliers

```
Number of outliers values = Length of all values - Length of all non
outliers values
```

In our code, `len(data)` determines *Length of all values and Length of all non outliers* values is determined by `len([x for x in trip_duration if q75+(1.5*iqr) >=x>= q25-(1.5*iqr)]))`.

Hence then the formula in Listing 1-20 was applied to calculate the ratio of values considered outliers.

Listing 1-21. Formula for Calculating Ratio of Outlier Values

```
Ratio of outliers = ( Number of outliers values / Length of all values ) * 100
```

Nancy was relieved to see only 9.5% of the values within the dataset to be outliers. Considering the time series nature of the dataset she knew that removing these outliers wouldn't be an option. Hence she knew that the only option she could rely on was to apply transformation to these outliers to negate their extreme nature. However, she was interested in observing the mean of the non-outlier values of trip duration. This she then wanted to compare with the mean of all values calculated earlier in Listing 1-15.

Listing 1-22. Calculating z scores for Observations Lying Within tripduration

```
mean_trip_duration = np.mean([x for x in trip_duration if q75+(1.5*iqr)
>=x>= q25-(1.5*iqr)])
upper_whisker = q75+(1.5*iqr)
print 'Mean of trip duration: %f'%mean_trip_duration
```

Output

```
Mean of trip duration: 711.726573
```

The mean of non-outlier trip duration values in Listing 1-22 (i.e., approximately 712) is considerably lower than that calculated in the presence of outliers in Listing 1-15 (i.e., approximately 1,203). This best describes the notion that mean is highly affected by the presence of outliers in the dataset.

Nancy was curious as to why Eric initialized the variable `upper_whisker` given that it is not used anywhere in the code in Listing 1-22. Eric had a disclaimer for this: "upper_whisker is the maximum value of the right (i.e., positive) whisker i.e. boundary uptill which all values are valid and any value greater than that is considered as an outlier. You will soon understand why we initialized it over here."

Eric was interested to see the outcome statistics once the outliers were transformed into valid value sets. Hence he decided to start with a simple outlier transformation to the mean of valid values calculated in Listing 1-22.

Listing 1-23. Calculating Mean Scores for Observations Lying Within tripduration

```
def transform_tripduration(x):

    if x > upper_whisker:
        return mean_trip_duration
    return x

data['tripduration_mean'] = data['tripduration'].apply(lambda x: transform_
tripduration(x))

data['tripduration_mean'].plot.hist(bins=100, title='Frequency distribution
of mean transformed Trip duration')
plt.show()
```

Eric remembers walking Nancy through the code in Listing 1-23.

> *We initialized a function by the name of `transform_tripduration`. The function will check if a trip duration value is greater than the upper whisker boundary value, and if that is the case it will replace it with the mean. Next we add `tripduration_mean` as a new column to the data frame. We did so by custom modifying the already existing `tripduration` column by applying the `transform_tripduration` function.*

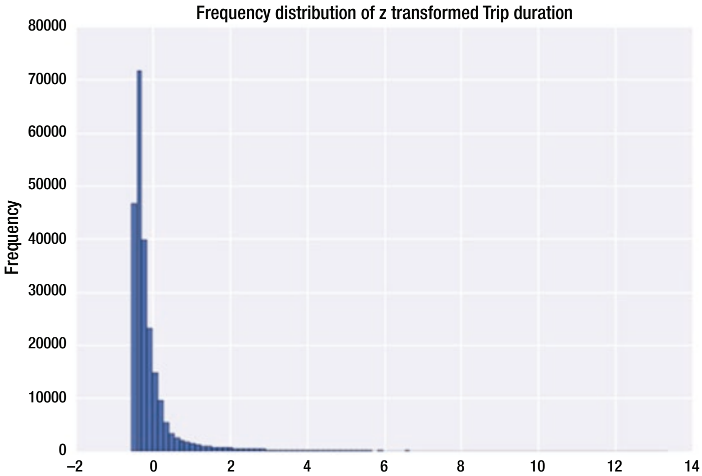
Nancy was of the opinion that the transformed distribution in Figure 1-16 is a positively skewed normal distribution. Comparing Figure 1-16 to Figure 1-10 reveals that the skewness has now decreased to a great extent after the transformation. Moreover, the majority of the observations have a `tripduration` of 712 primarily because all values greater than the upper whisker boundary are not converted into the mean of the non-outlier values calculated in Listing 1-22. Nancy was now interested in understanding how the center of measures appear for this transformed distribution. Hence Eric came up with the code in Listing 1-24.

Listing 1-24. Determining the Measures of Center in Absence of Outliers

```
print 'Mean of trip duration: %f'%data['tripduration_mean'].mean()
print 'Standard deviation of trip duration: %f'%data['tripduration_mean'].std()
print 'Median of trip duration: %f'%data['tripduration_mean'].median()
```

Output

```
Mean of trip duration: 711.726573
Standard deviation of trip duration: 435.517297
Median of trip duration: 633.235000
```



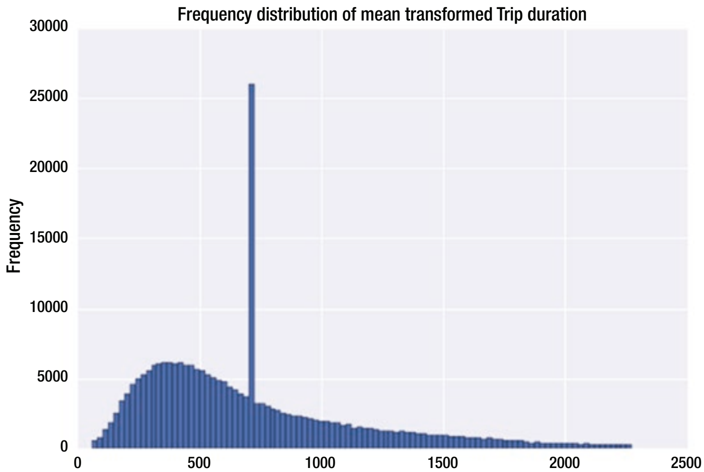


Figure 1-16. *Frequency distribution of mean transformed trip duration*

Nancy was expecting the mean to appear the same as that in Listing 1-22 because of the mean transformation of the outlier values. In Figure 1-16 she knew that the hike at 711.7 is the mode, which meant that after the transformation the mean is the same as that of the mode. The thing that surprised her the most was that the median is approaching the mean, which means that the positive skewness we saw in Figure 1-16 is not that strong.

On the basis of the findings in Figure 1-1, Nancy knew that males dominate females in terms of trips taken. She was hence interested to see the trip duration of males and repeat the outlier treatment for them as well. Hence she came up with these exercise questions for you in the hopes of gaining further insights.

EXERCISES

1. Find the mean, median, and mode of the trip duration of gender type male.
2. By looking at the numbers obtained earlier, in your opinion is the distribution symmetric or skewed? If skewed, then is is it positively skewed or negatively skewed?
3. Plot a frequency distribution of trip duration for trips availed by gender type male. Does it validate your inference as you did so in the previous question?
4. Plot a box plot of the trip duration of trips taken by males. Do you think any outliers exist?
5. Apply the formula in Listing 1-6 to determine the percentage of observations for which outliers exists.
6. Perform the treatment of outliers by incorporating one of the methods we discussed earlier for the treatment of outliers.

The multivariate analysis that Nancy and Eric had performed had yielded some good insights. However, Nancy was curious to know if some statistical tests exist to determine the strength of the relationship between two variables. She wanted to use this information to determine the features which have the most impact on trip duration. The concept of correlation popped up in Eric's mind, and he decided to share his knowledge base before moving on further with the analysis.

Correlation

Correlation refers to the strength and direction of the relationship between two quantitative features. A correlation value of 1 means strong correlation in the positive direction, whereas a correlation value of -1 means a strong correlation in the negative direction. A value of 0 means no correlation between the quantitative features. Please note that correlation doesn't imply causation; that is, the change in one entity doesn't enforce a change in the other one.

Correlation of an attribute to itself will imply a correlation value of 1. Many machine learning algorithms fail to provide optimum performance because of the presence of multicollinearity. Multicollinearity refers to the presence of correlations among the features of choice, and thus it is usually recommended to review all pair-wise correlations among the features of a dataset before considering them for analysis.

Following are the most common types of correlations:

Pearson R Correlation

Pearson R correlation is the most common of the three and is usually suitable to calculate the relationships between two quantitative variables which are linearly related and seem to be normally distributed. Take, for example, two securities in the stock market closely related to one another and examine the degree of relationship between them.

Kendall Rank Correlation

As compared to Pearson, which is suitable for normally distributed data, Kendall rank correlation is a non-parametric test to determine the strength and direction of relationship between two quantitative features. Non-parametric techniques are targeted to distributions other than the normal distribution. To the contrary, parametric techniques are targeted toward normal distribution.

Spearman Rank Correlation

Spearman rank correlation is a non-parametric test just like Kendall rank correlation, with the difference that Spearman rank correlation does not make any assumptions about the distribution of the data. Spearman rank correlation is most suitable for ordinal data.

Nancy was interested to see if change in age brings a linear change to trip duration. For that Eric decided to bring Pearson R correlation into practice and decided to make a scatter plot between the two quantities for them to see the relationship visually.

Listing 1-25. Pairplot of trip duration and age

```
data = data.dropna()
seaborn.pairplot(data, vars=['age', 'tripduration'], kind='reg')
plt.show()
```

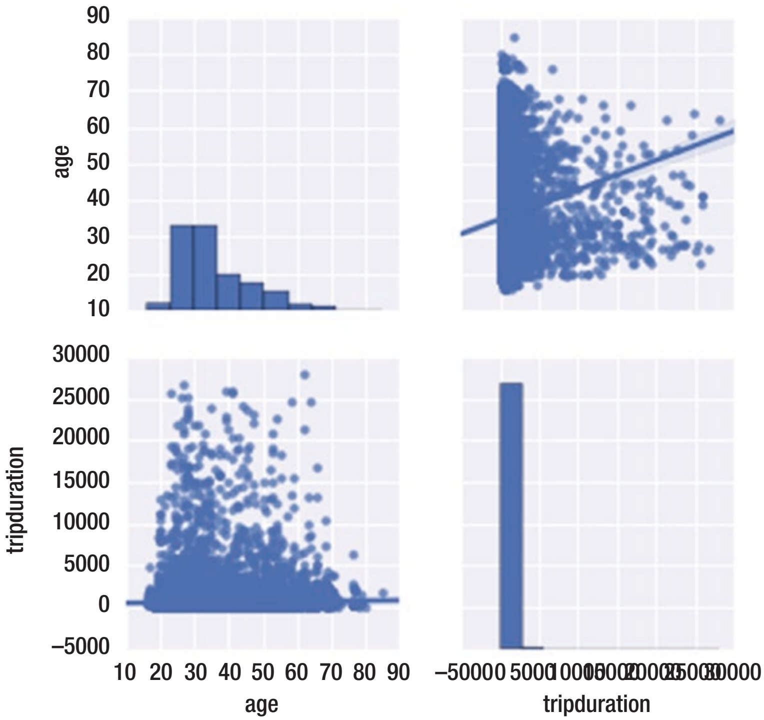


Figure 1-17. *Pairplot between trip duration and age*

While looking at Figure 1-17, Nancy didn't find any definitive pattern between trip duration and age. There is a minor positive correlation, as explained in Figure 1-18.

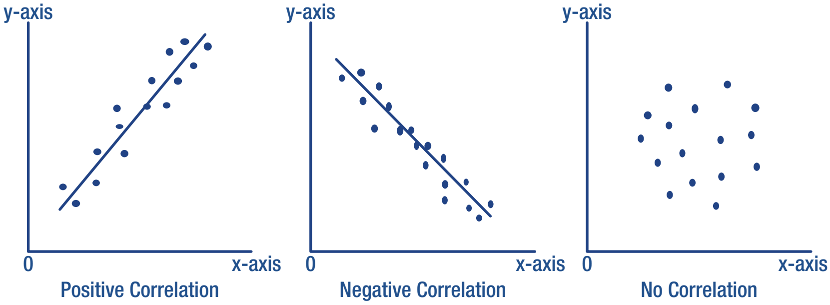


Figure 1-18. *Correlation directions*

Nancy knew that a perfect positive correlation meant a value of 1; hence she wanted to see if the correlation value between age and `tripduration` is positive and approaches 1 or not. Eric wrote the code in Listing 1-26 to make it possible.

Listing 1-26. Correlation Coefficient Between trip duration and age

```
pd.set_option('display.width', 100)
pd.set_option('precision', 3)

data['age'] = data['starttime_year'] - data['birthyear']

correlations = data[['tripduration','age']].corr(method='pearson')
print(correlations)
```

Output

```
             tripduration    age
tripduration 1.000           0.058
age          0.058           1.000
```

The correlation coefficient came out to be greater than 0 which according to Nancy's deduction was a positive correlation, but being much less than 1 meant it to be weak in nature.

Nancy was aware that a simple analysis meant taking a feature into consideration and analyzing it. Another more complex method was to split the feature into its categories (e.g., splitting gender into male and female) and then performing the analysis on both these chunks separately. She was confused as to which was the right approach and thus asked Eric for his opinion. Eric thought of introducing the concept of t-statistics and came up with a small demonstration for Nancy.

Hypothesis Testing: Comparing Two Groups

Before going through the methodology to compare two groups, it is important to understand null and alternative hypotheses.

Null hypothesis is something we attempt to find evidence against in the hypothesis tests. Null hypothesis is usually an initial claim that researchers make on the basis of previous knowledge or experience. Alternative hypothesis has a population parameter value different from that of null hypothesis. Alternative hypothesis is something you hope to come out to be true. Statistical tests are performed to decide which of these holds true in a hypothesis test. If the experiment goes in favor of the null hypothesis then we say the experiment has failed in rejecting the null hypothesis. You might be thinking, “Ain’t it the same” as that of accepting the null hypothesis? Well, no! Not guilty doesn’t actually mean that a person is innocent.

t-Statistics

t-refers to a difference represented in the units of standard error. This exists in two most common variants:

1-sample-t

This is the search of evidence of a significant difference between a population mean and a hypothesized value.

2-sample-t

This is when you are trying to find the evidence of a significant difference between population means. Note that the samples from both the distributions should be independent of each other.

t-value and p-values are internally linked. The larger the absolute value of the t-value, the smaller the p-value as the distribution is concentrated around the mean, and thus the higher the probability to reject the null hypothesis.

t-value can be negative as well as positive. The greater the magnitude of T in either direction, the greater is the evidence against the notion that there is no significant difference (i.e., the null hypothesis). Vice versa, if the t-value approaches 0, then it implies that there is no significant difference between the population means or between a population mean and a hypothesized value.

Contrary to a statistical phenomenon like correlation, a t-value is calculated by sampling one value from the entire population. Given that the sampling is random, each time a t-value is calculated the answer presents a random sampling error (i.e., some variation from the t-value calculated earlier).

t-distributions help identify the magnitude of change due to random sampling from the entire population.

t-Distributions and Sample Size

Degree of freedom refers to the instances in which you have the freedom to choose from more than one choice. Usually this is sample size - 1. Degree of freedom (DF) defines the spread of a Gaussian distribution. The more the DF, the lesser probability density lies within the tails, with the distribution tightly clustered around the mean. Small sample sizes are more likely to be unusual and thus pose a threat of getting further away from the null hypothesis even though the null hypothesis holds true. Hence it is better to have a large sample size when calculating t-value.

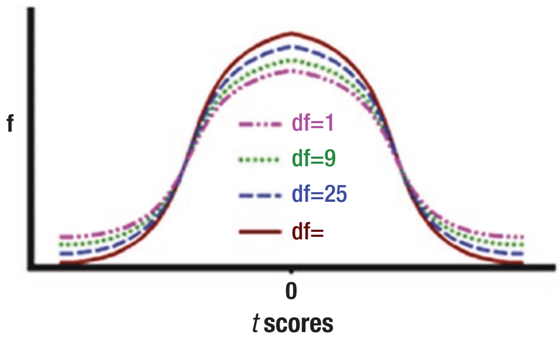


Figure 1-19. *Change in t-distributions due to changes in sample size*

Let's calculate the two-tail t-test for all categories of genders and user types (see Listing 1-27). We do this so we can understand if different categories have resemblance in variances. If they do, then they will be considered a group; otherwise they will be treated separately in the analysis mode.

Listing 1-27. Computing Two-Tail t-test of Categories of gender and user types

```
for cat in ['gender','usertype']:

    print 'Category: %s\n'%cat
    groupby_category = data.groupby(['starttime_date', cat])['starttime_
    date'].count().unstack(cat)
    groupby_category = groupby_category.dropna()
    category_names = list(groupby_category.columns)

    for comb in [(category_names[i],category_names[j]) for i in
    range(len(category_names)) for j in range(i+1, len(category_names))]:

        print '%s %s'%(comb[0], comb[1])
        t_statistics = stats.ttest_ind(list(groupby_category[comb[0]]),
        list(groupby_category[comb[1]]))
```

```
        print 'Statistic: %f, P value: %f'%(t_statistics.statistic,
        t_statistics.pvalue)
        print '\n'
```

Output

```
Category: gender

Female Male
Statistic: -38.572176, P value: 0.000000

Female Other
Statistic: 48.248666, P value: 0.000000

Male Other
Statistic: 53.180282, P value: 0.000000

Category: usertype

Member Short-Term Pass Holder
Statistic: 14.393456, P value: 0.000000
```

Neither the code nor the output made any sense to Nancy; hence Eric explained the following:

> *At first we looped over the two features gender and age. For each of these features, we first split its column into category columns. For example, gender was split into Male, Female, and Others. Next we made category pairs and ran t-statistics on them. For example, Male-Female, Male-Others, etcetera.*
>
> *The results seem to be homogeneous across all categories. What do I mean by that? Well for all of the comparisons, the p-values in the output seem to be roughly 0. If we go with a confidence interval of 95%, then it translates to a p-value of 0.05. None of the statistics above exceeds our set p-value benchmark. This leads us to the conclusion that we need to treat all of these categories separately when moving in the modeling aspect, as all of them have different variances.*

Nancy now knew her audience to a certain degree, but she wanted to know more, and for that purpose she planned on generating a question and doing sampling. Thus she asked Eric if he could enlighten her with a concept or two along those lines. Eric knew that central limit theorem has a pivotal value in sampling and analysis, so he came up with an illustration.

Central Limit Theorem

Consider as an example any distribution consisting of random samples with a well-defined mean and standard deviation. Now let's pick some random samples from that distribution and calculate the mean and standard deviation. As you increase the sample size (i.e., n), the mean and standard deviation converge to that of a Gaussian normal distribution. In other words, as you increase sampling from an arbitrary distribution it converges to a normal distribution with a given mean and standard deviation. So how large should the sampling size be for the theorem to influence the behavior?

- The greater the sample size is, the more closely that sampling distribution will resemble the normal distribution.
- The more the arbitrary distribution resembles the normal distribution, the fewer samplings will be required to satisfy the theorem. Statisticians usually recommend a sample size of 30 to 40 in a scenario like this one.

Eric decided to use the data at hand as an aid to explain central limit theorem (see Listing 1-28).

Listing 1-28. Script to Validate Central Limit Theorem on Trips Dataset

```
daily_tickets = list(data.groupby('starttime_date').size())
sample_tickets = []
checkpoints = [1, 10, 100, 300, 500, 1000]
plot_count = 1

random.shuffle(daily_tickets)

plt.figure(figsize=(15,7))
binrange=np.array(np.linspace(0,700,101))

for i in xrange(1000):
    if daily_tickets:
        sample_tickets.append(daily_tickets.pop())

    if i+1 in checkpoints or not daily_tickets:
        plt.subplot(2,3,plot_count)
        plt.hist(sample_tickets, binrange)
        plt.title('n=%d' % (i+1),fontsize=15)
        plot_count+=1

    if not daily_tickets:
       break

plt.show()
```

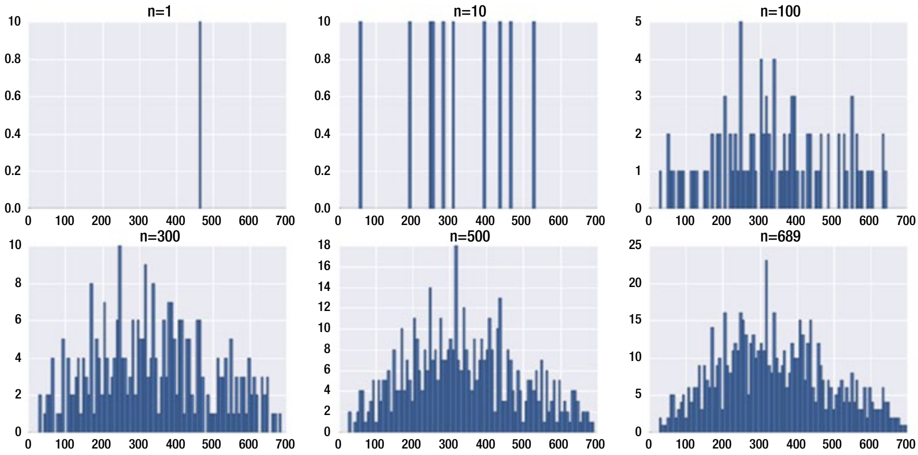


Figure 1-20. *Distributions of daily tickets on different sample sizes*

Eric explained the code in Listing 1-28 as follows:

> *Random shuffling was done in order to change the order of daily number of tickets. Once done, then different sample sizes were fixed in order to see the change in distribution as a function of the sample size. Note in Figure 1-20 that with the increase in sample size, the distribution seems to transform into a normal distribution. This validates the central limit theorem.*

Case Study Findings

Eric and Nancy's deductions helped them understand their audience to a better extent and garner valuables insights. The insights were rendered on data collected from 2014 to 2016 with demographic information only available for the members and not short-term pass holders. Hence, in order to get information about the short-term pass holders Nancy knew that she had to go through a market research exercise where central limit theorem would come in handy. Trip duration follows a definite seasonal pattern that repeats over time. Forecasting this time series can help Nancy predict the times when the company needs to push its marketing efforts and times when most trips anticipated can help ensure operational efficiencies. As for the promotions, Nancy now knew that the best station at which to kick off the campaign would be Pier 69/Alaskan Way & Clay St. Outliers were a tiny portion of the dataset; however, their time series nature meant that those outliers couldn't be removed and transformation was thus applied. Regarding further analysis, Nancy was now aware that as the features are not homogeneous, the analysis would have to be done on the individual category level.

Nancy wasn't sure if the techniques they applied were obsolete or have applications in real world applications as well. Hence Eric compiled the following list of applications that his friends from the industry use to bring Statistics and Probability into live.

Applications of Statistics and Probability

Applications of statistics and probability are vibrant in several fields of study.

Actuarial Science

Actuaries use the concepts of mathematics, probability theory, statistics, finance, economics, computer science, and modeling to evaluate and price risks. Their application cases exist in the domains of insurance, consulting firms, and government.

Biostatistics

There are applications of statistics in various branches of biology. This encompasses the design of biological experiments and making inferences from them. Diving deeper into biostatistics reveals examples in which subjects (patients, cells, etc.) exhibit variation in response to some stimuli (e.g., medicine). Biostatisticians use inferential statistics to give meaning to these anomalies.

Astrostatistics

Astrostatistics is an amalgam of statistical analysis, astrophysics, and data mining. Data collected from automatic scanning of cosmos is used to make deductions.

Business Analytics

Business analytics uses operational and statistical theories to make predictive models. It also incorporates optimization techniques to garner effective insights for customers and business executives.

These insights enable companies to automate and optimize their business processes. Business intelligence differs from business analytics in that business intelligence helps us answer what happened whereas business analytics helps us understand the reason for this anomaly (i.e., why it happened in the first place and the chances of it happening again). These analytics are used in various business areas such as enterprise optimization, fraud analytics, pricing analytics, supply chain analytics, and so on.

Econometrics

The application of statistical methods for estimating economic relationships constitutes econometrics. Some of the examples include measuring the effect of divorce laws on divorce and marriage rates, change in wages of native workers from impact on immigration policies, or forecasting macroeconomic variables of inflation rates, interest rates, or gross domestic product.

Machine Learning

Several machine learning algorithms are based on statistical theories or an advanced version of the same. An example of this is the Bayesian theory which is commonly used.

Statistical Signal Processing

Past corpus of speeches is used to determine the highest probability of spoken words. Moreover, statistical signal processing is used in the following applications:

- Game theory
- Estimation and filtering
- Signal processing
- Linear systems

Elections

Campaign managers use the results of the polls to infer wins in the coming elections for their political parties.

CHAPTER 2

Regression

Regression and time series analysis make predictions for quantitative target variables possible. This chapter aims to highlight the core concept of regression and its variants. The emphasis here is to take the readers through the journey of model selection when solving for real-world problems. Moreover, this chapter also features statistical tests to evaluate the findings of these regression techniques.

Note This book incorporates **Python 2.7.11** as the de facto standard for coding examples. Moreover, you are required to have it installed for the *Exercises*.

In this chapter we will be using the **Concrete Compressive Strength dataset** for coding examples and exercises. This data dump can be downloaded from UCI's web site link:

http://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

Case Study: Removing Inconsistencies in Concrete Compressive Strength

Andrew had an hour-long meeting with Smith regarding the frequent complaints he had been receiving from their existing clientele. Smith was the sales representative at a construction aggregates company and was facing difficulties in meeting his sales targets. Smith believed that the reason for his difficulties was that the existing clients had complaints regarding quality inconsistencies—this, when coupled with negative word of mouth in the market, was making it tough for him to seal deals with potential customers.

Andrew, the plant supervisor, was Smith's friend. At first Smith tried to find a solution for this problem, but when things got out of his control, he decided to have a meeting with Andrew.

D. Haroon, *Python Machine Learning Case Studies*, DOI 10.1007/978-1-4842-2823-4_2

There is usually friction between the production and sales departments in most companies. Both of them usually fight over their yearly budgets and their plans for meeting their annual targets. Smith knew this but he felt sure that things would be different with Andrew as they had been friends for some time. However, as the meeting kicked off and Smith presented the matter to Andrew, to his dismay Andrew wasn't ready to acknowledge any inconsistencies in production. As Smith recalled Andrew saying, "These complaints don't make any sense to me. You ask why! Well first of all, we had an upgrade at our production plant in the last quarter, and you know for a fact that the raw material we use goes through a thorough testing prior to being used for production."

The meeting didn't help Smith in any way, and he let the matter go for some time. One day, when he was in a networking session during one of his company's corporate events, Smith heard some executives from the production department talking about the recent upgrade of the plant. He recalled hearing one of the executives say, "Though we don't have any testing improvement in our recent upgrade for measuring concrete compressive strength, then too our product quality is superior thanks to our robust testing of raw material used."

When Smith heard this, he did some research and was surprised to discover that the complaints had gained momentum ever since the upgrade had taken place. However, he also discovered that the procurement of the testing equipment was put on hold due to a shortage of funds allocated to the procurement department. Smith knew for a fact that he couldn't afford to wait until the next year for the issue to resolve on its own, and he had to find a way to make things right.

He had heard that the recent upgrade meant that machines had recording data. Smith decided to set up a meeting with Claire (the manager of analytics) to see if she could help in figuring out the reason behind the inconsistencies in concrete compressive strength. The meeting went well for Smith, as Claire assured him of the in-house analytics capabilities. As Claire recalled, "At that time we had started up the analytics vertical, but I knew for sure that we had the muscles to formulate a model which can answer this anomaly."

Smith was relieved and now had to formulate a strategy to take his findings forward—that is, either approach management with his findings or share his findings to the production department for them to integrate those into their inner processes. However, Smith was curious if the data at hand was strong enough to deduce some quality findings. Moreover, he was interested to see which factors influence concrete compressive strength the most.

Smith was interested to know which features of the dataset Claire would be working on. Hence, Claire came up with the data dictionary in Table 2-1.

Table 2-1. *Data Dictionary for the Concrete Compressive Strength Dataset*

Feature name	Description
Cement (kg in a m3 mixture)	Amount of cement used in a m3 mixture (unit: kg)
Blast furnace slag (kg in a m3 mixture)	Amount of blast furnace slag used in a m3 mixture (unit: kg)
Fly ash (kg in a m3 mixture)	Amount of blast fly ash usedin a m3 mixture (unit: kg)
Water (kg in a m3 mixture)	Amount of water used in a m3 mixture (unit:kg)
Superplasticizer (kg in a m3 mixture)	Amount of superplasticizer used in a m3 mixture (unit: kg)
Coarse aggregate (kg in a m3 mixture)	Amount of coarse aggregate used in a m3 mixture (unit: kg)
Fine aggregate (kg in a m3 mixture)	Amount of fine aggregate used in a m3 mixture (unit: kg)
Age (days)	Age of concrete (unit: days)
Concrete compressive strength	Concrete compressive strength which is measured in MegaPascal (MPa). This is the unit for pressure or stress and is the common unit to determine compressive strength of concrete.

Before moving forward Claire thought of initializing the following packages. She preferred to do this in order to avoid bottlenecks while implementing the code snippets on her local machine (Listing 2-1).

Listing 2-1. Importing Packages Required for This Chapter

```
%matplotlib inline

import time
import random
import datetime
import pandas as pd
import matplotlib.pyplot as plt
import statistics
import numpy as np
from scipy import stats
from sklearn.grid_search import GridSearchCV
from sklearn.linear_model import RANSACRegressor, LinearRegression,
TheilSenRegressor
```

```
from sklearn.metrics import explained_variance_score, mean_absolute_error,
mean_squared_error, median_absolute_error, r2_score

from sklearn.svm import SVR
from sklearn.linear_model import Ridge,Lasso,ElasticNet,BayesianRidge
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.cross_validation import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.cross_validation import cross_val_score
import seaborn
from IPython.display import Image
```

Smith was curious as to which techniques Claire would use to solve this problem. Claire thought this problem to be the ideal application for regression as the response variable (concrete compressive strength) is a quantitative quantity, and she was concerned about finding the factors that influence concrete compressive strength. These influential factors, when scaled by their composition and added together, can produce the compressive stength of concrete. Thus this will enable the calculation of concrete compressive strength without the need to procure any equipment specifically for the problem at hand. To help Smith understand what regression is, she compiled information on the topic.

Concepts of Regression

Regression describes the relationship between an exploratory variable (i.e., independent) and a response variable (i.e., dependent). Exploratory variables are also referred to as predictors and can have a frequency of more than 1. Regression is being used within the realm of predictions and forecasting. Regression determines the change in response variable when one exploratory variable is varied while the other independent variables are kept constant. This is done to understand the relationship that each of those exploratory variables exhibits. Please note that irrespective of classification that is used to predict discrete response variables, regression is used to predict response variables that are continuous in nature. A basic variant of regression is linear regression.

Interpolation and Extrapolation

Extrapolation refers to the use of regression for predicting response variable values outside the range of exploratory values used initially for training the model. This data is not used for training the model and is represented by the dotted line in Figure 2-1.

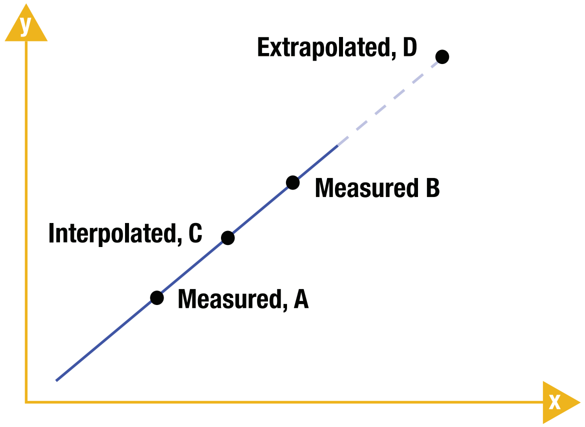


Figure 2-1. *Visual representation of interpolation and extrapolation*

The regression model will be fitted/trained on the data at hand (i.e., interpolation represented by the solid line) and the trained model will then be used to extrapolate on the portion represented by a dotted line. Hence, interpolation is done on data already known, and extrapolation is done on the data unknown.

Linear Regression

Linear regression is a form of regression in which one exploratory variable is used to predict the outcome of a response variable. Mathematically it can be defined as follows:

- Symbol: y
- Formula: $y = b_0 + b_1x + e$
- Illustration:
  - y: *Response variable*
  - x: *Explanatory variable*
  - b_0: *Slope*
  - b_1: *Intercept*
  - e: *Regression residual*

Figure 2-2 shows this equation visually.

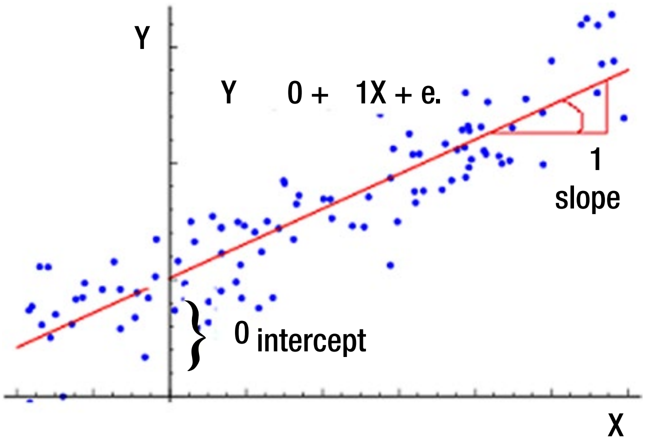


Figure 2-2. *Linear regression best fit line along with the regression coefficients explained*

Please note in Figure 2-2 that intercept is the point where the best fit line crosses the y axis. Regression residual is the difference between the actual value of the response variable and the predicted one.

Least Squares Regression Line of y on x

The least squares regression line makes the sum of square vertical distances of data points from the line as minimal as possible. Mathematically it can be defined as follows:

- Symbol: $\hat{y}$
- Formula: $\hat{y} = b_0 + b_1 x$
- Further derivations:
  - $b_1 = r\frac{s_y}{s_x}$
  - $b_0 = \acute{y} - b_1 \acute{x}$
- Least squares regression line always passes through $(\acute{x}, \acute{y})$

The illustration in Figure 2-3 uses linear regression (i.e., one exploratory variable to depict the methodology adapted by the least squares regression line). y depicts the response variable whereas x denotes the exploratory variable. Vertical distances from data points to the lines are shown and the best fit regression line is seen as passing through the center of these data points. Thus this ensures that the square of residual is minimized for each data point.

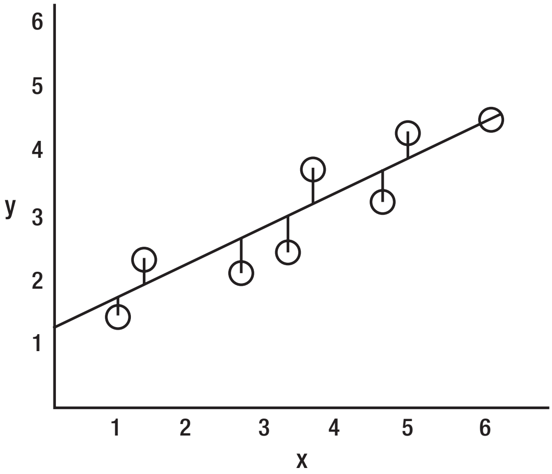


Figure 2-3. *Residuals*

Multiple Regression

Multiple regression is a type of regression in which more than one exploratory variable is used to predict the outcome of a response variable. Multiple regression is mathematically defined as follows:

- Symbol: y
- Formula: $y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \ldots.. b_tx_t + e$
- Illustration:
    - *y: Response variable*
    - *$x_1, x_2, x_3, \ldots\ldots, x_t$: Explanatory variable*
    - *b_0: Slope*
    - *$b_1, b_2, b_3, \ldots\ldots., b_t$: Intercepts*
    - *e: Regression residual*

The formula above can also be referred to as the equation of regression. Essentially, one is trying to find optimal values for all coefficients such can give the best possible value of y given all x. In the case of single variable, it's a line (i.e., linear regression), as shown in Figure 2-3, but in case of multiple variables it becomes a plane, as shown in Figure 2-4.

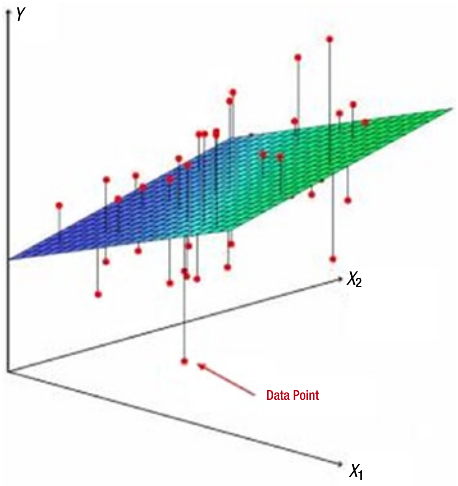


Figure 2-4. *Multiple regression with two exploratory and one response variables explained*

Plugging optimal values of coefficients (i.e., slope and intercepts) back in the formula above will yield us an equation. Unknown values in this equation will be the x values, which are the exploratory variable values in this case. For any given value of x, the equation will return us with a prediction.

Stepwise Regression

This type of regression is a customized version of multiple regression. Rather than using all exploratory variables for predicting the outcome, stepwise regression automatically selects the independent variables from a given pool which can yield a best fit line. This can be done by doing either of the following:

- Try out one exploratory variable at a time and use that for fitting the regression model. Incorporate that exploratory variable if found statistically significant.
- Run multiple regression on all exploratory variables and use the one that is most statically significant to determine the best fitted regression line.

Figure 2-5 best describes the methodology adopted by stepwise regression.

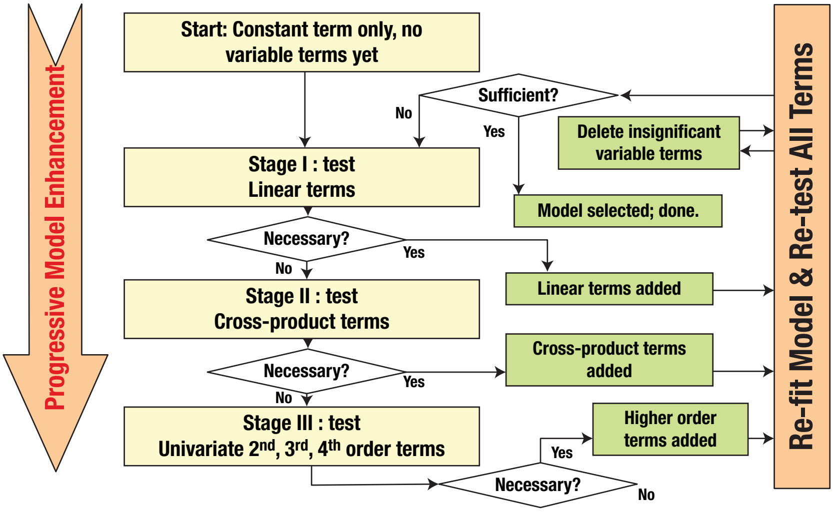


Figure 2-5. *Flowchart of stepwise regression Source:.slideshare.net/bzinchenko/quant-trader-algorithms*

However, as per some statisticians, this approach inherits the following problems:

- Opting for one exploratory variable and dropping others creates a bias toward the one selected and deprives one from observing the chunk of variations which could have been captured by the exploratory variables dropped in the first place.
- In the case of choosing among many exploratory variables, going through this entire exercise will require significant computing power, and thus time.

Polynomial Regression

Sometimes when plotting the scatter plot between exploratory and response variables we see a non-linear trend. This trend will go unnoticed by the linear regression cameo, and will thus require a non-linear treatment where polynomial regression comes in handy. Polynomial regression uses degrees of polynomial to make a non-linear regression line. Polynomial regression can mathematically be defined as follows:

- Symbol: y
- Formula: $y = b_0 + b_1x + b_2x^2 + b_3x^3 + b_hx^h + e$
- Illustration:
  - *y: Response variable*
  - *$x_1, x_2, x_3, \ldots\ldots, x_h$: Explanatory variable*

- b_0: *Slope*
- $b_1, b_2, b_3, \ldots\ldots, b_h$: *Intercepts*
- e: *Regression residual*
- h: *Degree of polynomial*

Looking at Figure 2-6 you will notice that increasing the degree of polynomial makes the curve more non-linear and induces additional curves within the regression line.

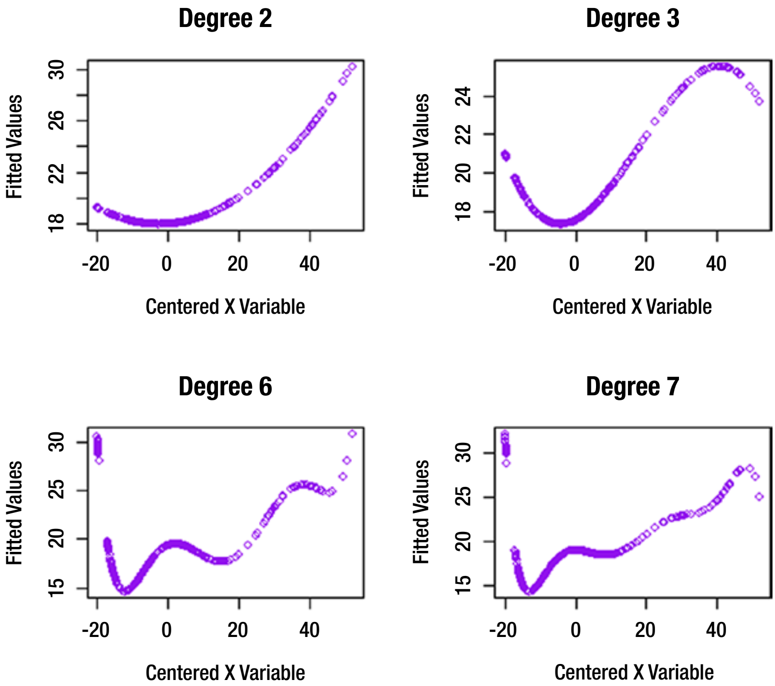


Figure 2-6. *Polynomial regression curves for different values of degrees (i.e., h)*

Assumptions of Regressions

In order to prevent Smith from thinking of regression as the magic wand for continuous data predictions, Claire came up with the set of assumptions that should hold true for regression to be applicable to a given dataset.

Number of Cases

The ratio of cases-to-independent variables (IVs) should ideally be 20:1. This means that there should be 20 samples for each exploratory variable. However, in extreme cases a minimum of 5 samples to one exploratory variable is permissible (i.e., for 5 exploratory variables there should be 25 samples in the dataset).

Missing Data

In regression, missing data can lead to a model which is unfit for the sampling data. You might fall victim to either one of the following situations:

- There might be instances when some observations have missing values for all fields within the data. Removing them will be a wise thing to do because if the missing values are insignificant then neglecting them won't disturb the overall behavior captured by the model.
- In case only a given column has missing values in majority of the observations no treatment is required because then regression will neglect the cases in which there are no values for that variable.

Outliers

Outliers can create a bias in the outcome of a regression model such that they will pull the best fit line toward themselves. Hence treatment of outliers is critical in regression and can be done by means of an appropriate method selected from the ones highlighted in Chapter 1.

Multicollinearity and Singularity

Multicollinearity and Singularity are two concepts which undermines the regression modeling, resulting in bizarre and inaccurate results. If exploratory variables are highly correlated, then regression becomes vulnerable to biases. Multicollinearity refers to a correlation of 0.9 or higher, whereas singularity refers to a perfect correlation (i.e., 1).

A remedy to this is to remove the exploratory variable exhibiting correlation of more than 0.7. But then this brings us to the following problem. Considering the example in Figure 2-7, where weight and blood pressure (BP) have a high correlation of 0.95.

Correlation: BP, Age, Weight, BSA, Dur, Pulse, Stress

	BP	Age	Weight	BSA	Dur	Pulse
Age	0.659					
Weight	0.950	0.407				
BSA	0.866	0.378	0.875			
Dur	0.293	0.344	0.201	0.131		
Pulse	0.721	0.619	0.659	0.465	0.402	
Stress	0.164	0.368	0.034	0.018	0.312	0.506

Figure 2-7. *Illustration of multicollinearity*

Now which one (weight or BP) is the troublemaker and should be removed? This is the point where tolerance comes in as a remedy? Tolerance can be mathematically defined as follows:

- Formula: ***Tolerance* = *1-r²***
- Illustration:
    - ***r²****: Squared multiple correlation of this variable with all other independent variables in the regression equation*

Tolerance refers to the proportion of the exploratory variable's variance not captured by other exploratory variables in the equation. The greater this value, the better it is. Eradicating multicollinearity and singularity will ensure that the response variable is predicted from the variations captured by exploratory variables independently and not by the correlations which exist among them.

After reading through the description of the regression techniques available, Smith was curious to know which method would work best for the problem at hand. However, Claire knew that the process for figuring out the right method would need some understanding of the data before moving forward. Hence she decided to perform features' exploration. By features she meant variables within the dataset. She was primarily interested to see what the data looks like and if the following exist:

- Correlation between the exploratory variables and the response variable or not
- Multicollinearity and singularity

Features' Exploration

Claire started off by loading the data into memory (see Listing 2-2).

Listing 2-2. Reading the Data in the Memory

```
data = pd.read_csv('examples/concrete_data.csv')
```

Smith was curious to know how much data there was and what it looked like. Hence Claire wrote the code in Listing 2-3 to print some initial observations of the dataset to get a feel of what it contains.

Listing 2-3. Printing the Size of the Dataset and Printing the First Few Rows of the Dataset

```
print len(data)
data.head()
```

Output

```
1030
```

Table 2-2. *Print of Observations of the Dataset*

	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5) (kg in a m^3 mixture)	Coarse Aggregate (component 6) (kg in a m^3 mixture)	Fine Aggregate (component 7) (kg in a m^3 mixture)	Age (day)	Concrete compressive strength(MPa, megapascals)
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

While looking at the first few observations in Table 2-2, Smith noticed that all the features except that of age are floating point numbers, with age being the only integer variable. He also noticed that blast furnace slag, fly ash, and superplasticizer are not always the prerequisite when building a concrete. Moreover, he noticed that the concrete data within the dataset can be a year old (i.e., row no. 3 has an age of 365 ~ year)

Claire knew for sure that for the analysis (i.e., correlation, regression) to be done it is important that feature names should be as simple as possible. Hence she wrote the code snippet in Listing 2-4 to make the names brief and human readable. For renaming she followed the name mapping as defined in Table 2-3.

Listing 2-4. Renaming the Columns

```
data.columns = ['cement_component', 'furnace_slag', 'flay_ash',
'water_component', 'superplasticizer', \
            'coarse_aggregate', 'fine_aggregate', 'age', 'concrete_strength']
```

Table 2-3. *Variable Names' Mapping*

Old Feature name	New feature name
Cement(component 1)(kg in a m3 mixture)	cement_component
Blast Furnace Slag(component 2)(kg in a m3 mixture)	furnace_slag
Fly Ash(component 3)(kg in a m3 mixture)	flay_ash
Water(component 4)(kg in a m3 mixture)	water_component
Superplasticizer(component 5)(kg in a m3 mixture)	superplasticizer
Coarse Aggregate(component 6)(kg in a m3 mixture)	coarse_aggregate
Fine Aggregate(component 7)(kg in a m3 mixture)	fine_aggregate
Age (days)	age
Concrete compressive strength(MPa, megapascals)	concrete_strength

Having seen what the data looks like, it was time for Claire to see how well the exploratory variables correlate to the response variable and if any multicollinearity/singularity in the data exists or not.

Correlation

Claire first decided to see whether any of the exploratory variables correlates to the response variable (i.e., concrete strength). This question was of the utmost importance to her because she believed that high correlation between two quantitative entities can lead to a better best fit linear regression line. Hence she was interested to determine the strength and direction of relationships between these quantitative quantities. For that she wrote the code in Listing 2-5. However, before moving forward she explained why correlation and regression are not entirely the same:

> *Correlation can be calculated between any two quantitative quantities. However, regression is always commuted between a response variable and exploratory variable(s). Correlation is limited to just two quantitative quantities where regression can have more than two quantitative quantities, that is, one response variable and more than one exploratory variables aka multiple regression.*

Claire also pointed out that an easy way to see the correlation visually is to use a scatter plot between response variable and exploratory variables, one at a time (Figure 2-8).

Listing 2-5. Plotting Scatter Plots Between the Response and Exploratory Variables

```
plt.figure(figsize=(15,10.5))
plot_count = 1

for feature in list(data.columns)[:-1]:
        plt.subplot(3,3,plot_count)
        plt.scatter(data[feature], data['concrete_strength'])
        plt.xlabel(feature.replace('_',' ').title())
        plt.ylabel('Concrete strength')
        plot_count+=1

plt.show()
```

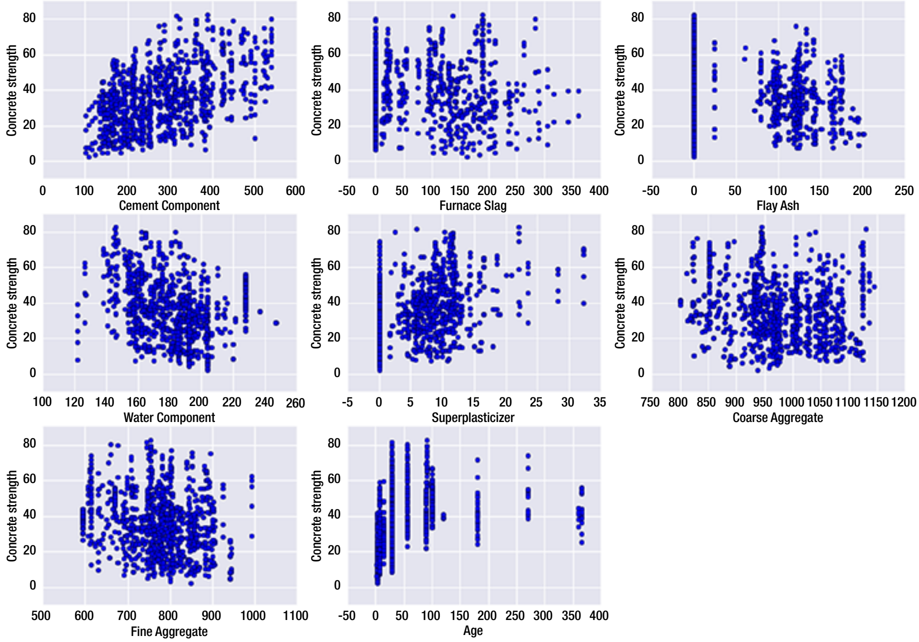


Figure 2-8. *Scatter plot between response variable and exploratory variables*

Claire came up with the description of the code in Listing 2-5 to make Smith at ease with the methodology used. She recalled saying the following:

> *We came up with multiples plots within the same figure by using subplots. We first defined the figure size by means of `plt.figure (figsize=(15,10.5))`. This fixes the figure to a width of 15 and height of 10.5. The response variable (that is, concrete strength) i.e. response variable remains the same for all plots, whereas exploratory variables are taken into consideration one at a time (that is, Plot by plot). Thus we used: `for feature in list(data.columns)[:-1]` which looped through all the variables except the last one (that is, the response variable). Then we defined the subplot index as follows: `plt.subplot (3,3,plot_count)`. The first three define the number of rows, whereas the other three define the number of columns. This means that the figure will be able to accommodate a maximum of 9 plots. `plot_count` defines the index where that specific plot will be positioned on the figure. Possible values within this scenario can be from 1 to 9.*

Smith decided to take a shot and explain the correlations by looking at Figure 2-8. Hence he came up with the following three insights:

- The presence of outliers is negligible in the majority of these plots except for the plot between concrete strength and age.
- In some of the scatter plots we see a high frequency of values lying on 0. This can be seen in the plots between the concrete `strength` variable and exploratory variables, primarily age, furnace slag, fly ash, and superplasticizer.
- There seems to exist instances in which positive, sometimes negative, and sometimes no correlation exists between the two quantitative quantities.

Claire decided to extend these findings by listing the pairs for each of the instances (i.e., no, positive and negative correlations). She did so by evaluating plots in Figure 2-6 on the benchmark correlation in Figure 2-9. In case there is a trend, it will be either a negative or a positive correlation. An absence of a definite trend will indicate the existence of no correlation.

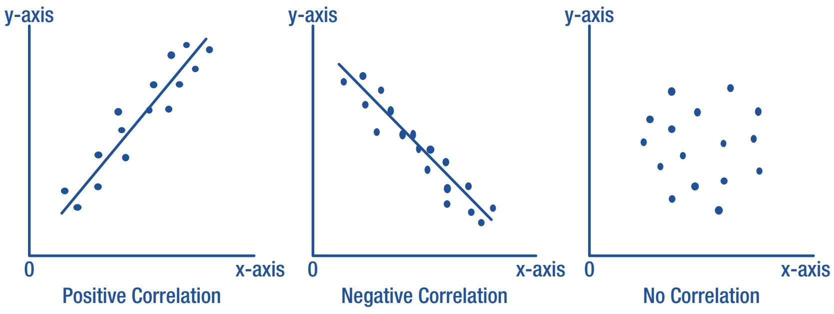


Figure 2-9. *Illustration of positive, negative, and no correlation*

- **Positive correlation** exists between
  - Cement component and concrete strength
  - Superplasticizer and concrete strength
- **Negative correlation** lies between
  - Fly ash and concrete strength
  - Water component and concrete strength
  - Coarse aggregate and concrete strength
  - Fine aggregate and concrete strength
- **No correlation** exists between
  - Furnace slag and concrete strength
  - Age and concrete strength

The pairs identified by the visual representation made it easy for Smith to understand what correlation is. However, he wanted to validate if the findings made sense statistically as well. Hence, Claire came up with the code snippet and computed Pearson correlation in Listing 2-6.

Listing 2-6. Calculating Pair-wise Pearson Correlations

```
pd.set_option('display.width', 100)
pd.set_option('precision', 3)
correlations = data.corr(method='pearson')
print(correlations)
```

	cement_component	furnance_slag	flay_ash	water_component	superplasticizer \
cement_component	1.000	-0.275	-0.397	-0.082	0.092
furnance_slag	-0.275	1.000	-0.324	0.107	0.043
flay_ash	-0.397	-0.324	1.000	-0.257	0.378
water_component	-0.082	0.107	-0.257	1.000	-0.658
superplasticizer	0.092	0.043	0.378	-0.658	1.000
coarse_aggregate	-0.109	-0.284	-0.010	-0.182	-0.266
fine_aggregate	-0.223	-0.282	0.079	-0.451	0.223
age	0.082	-0.044	-0.154	0.278	-0.193
concrete_strength	0.498	0.135	-0.106	-0.290	0.366

	coarse_aggregate	fine_aggregate	age	concrete_strength
cement_component	-0.109	-0.223	0.082	0.498
furnance_slag	-0.284	-0.282	-0.044	0.135
flay_ash	-0.010	-0.079	-0.154	-0.106
water_component	-0.182	-0.451	0.278	-0.290
superplasticizer	-0.266	0.223	-0.193	0.366
coarse_aggregate	1.000	-0.178	-0.003	-0.165
fine_aggregate	-0.178	1.000	-0.156	-0.167
age	-0.003	-0.156	1.000	0.329
concrete_strength	-0.165	-0.167	0.329	1.000

Figure 2-10. *Correlations between response variable and exploratory variables*

Claire recalled explaining the findings in the following words:

> *Output in Figure 2-10 shows a grid in which the Pearson correlation is computed between all features in the dataset and not confined to just the correlations between exploratory variables and response variable as we saw earlier in Figure 2-8. A strong positive correlation has a value of 1, strong negative correlation has a value of -1, and 0 indicates no correlation. By looking at the scatter plot in Figure 2-8 we had assumed that there exists no correlation between age and concrete strength; however, the statistical results in Figure 2-10 say otherwise; that is, there exists a slight positive correlation between the two quantities. Visual deduction is prone to human error so we will go with the latter (that is, there's a slight positive correlation between the two quantities).*

Smith, knowing how each exploratory variable correlates to the response variable, was curious to investigate whether or not singularity or multicollinearity exists in the dataset. This was exactly the next thing Claire had on list, and thus she wrote the code in Listing 2-7.

Listing 2-7. Calculating Pair Plot Between All Features

```
data_ = data[(data.T != 0).any()]
seaborn.        pairplot(data_, vars=data.columns, kind='reg')
plt.show()
```

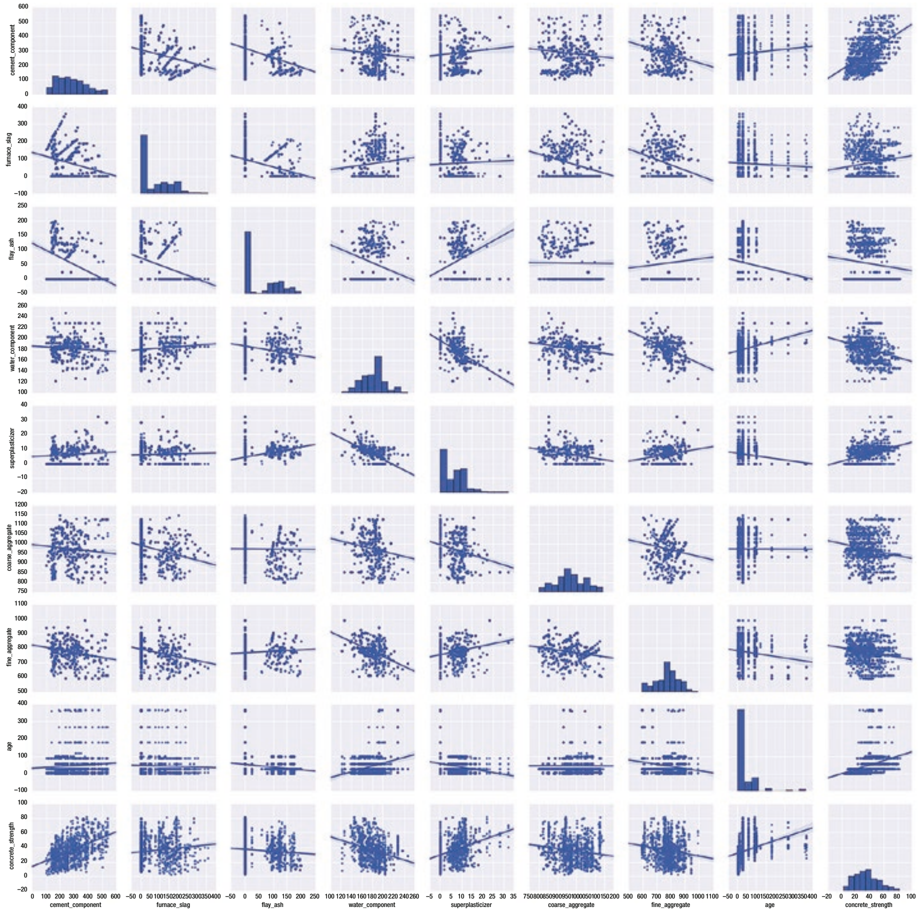


Figure 2-11. *Pair plot of all features in the dataset*

Claire decided to first explain to Smith what the code meant in Listing 2-7. She pointed out an aspect obvious in Figure 2-8 (i.e., that some of the features in the dataset have value of 0 in majority). She believed that having a majority of 0 values in the dataset can lead to correlation coefficients and regression lines that do not cover the dataset in its true essence. She started off by explaining that `data_= data[(data.T != 0).any()]` meant deleting the records in columns having a value of 0. After doing that she used seaborn pairplot to plot correlations between all features in the dataset.

While explaining the output in Figure 2-11, Claire asked Smith to concentrate on column 1 and row 2. Claire recalled to having explained that: "In many of the cases feature pairs don't have a correlation between them. Exception lies in two cases—that is, cement component and furnace slag and cement component and fly ash—where we can see a strong negative correlation."

Smith was overwhelmed by the insights he had gained so far. He had seen what the plots looked like after zero was removed from the observations. He was curious to

determine how the correlation coefficients would look after zeros are removed from the dataset. Moreover, he was aware that a Spearman correlation exists between these features hence he was curious to see what correlation coefficients it would present in the given situation. Finally, while looking at Figure 2-8, it seemed to Smith that all features were continuous except age, which looked like a discrete variable to him as observations within age fell within one of the seven discrete values, Smith was interested in splitting the data in to age segments and compute the correlations for each of these splits to see a strong correlation between the response variable and any of the exploratory variables. Can you help Smith answer these questions? Give it a shot in the following exercises.

EXERCISES

1. Remove zeros from the data features and then recalculate the Pearson correlation. Any improvements in the correlation coefficient?
2. Determine Spearman correlation for features in the dataset. Did this bring any marginal difference in the correlation scores?
3. From the scatter plot it seems that age is more of a discrete feature. Take the data from each age one by one and calculate the Pearson correlations for the remaining features. Does any specific age yield a good correlation between the response variable and exploratory ones?

Now that Claire and Smith understood the exploratory variables that had a significant influence on concrete strength, and with knowledge of where multicollinearity possibly exists, it was time to bring regression into practice. Regression was required for them to come up with an equation which can enable them to measure the concrete strength without the need of procuring some special equipment to measure it. However, before proceeding with regression models Claire felt she needed to make Smith aware of the important concepts of overfitting and underfitting.

Overfitting and Underfitting

Overfitting refers to the phenomenon where a model is highly fitted on a dataset. This generalization thus deprives the model from making highly accurate predictions about unseen data. Overfitting is shown by the non-linear line in Figure 2-12.

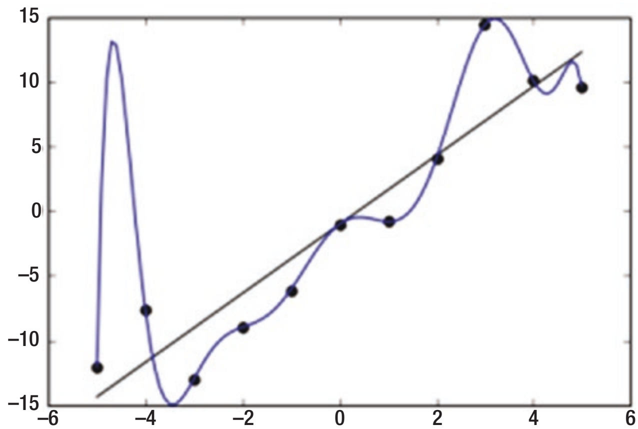


Figure 2-12. *Illustration of overfitting*

The following methodologies can be used to curb overfitting:

- Make models simple; that is, tune as many minimum parameters as possible, because the more complex they are, the more they will overfit the data at hand.
- Perform cross-validation. For example, randomly choose x% of the data values for train and the remaining y% for test. Fit the model on the train data, use it to predict values on the test data, and compute the test error. Repeat this exercise again n times. Each of the times the train/test split will be done randomly. Compute the average of all test errors to find out the real test error of the model at hand.

Underfitting is a phenomenon where the model is not trained with high precision on data at hand.

The treatment of underfitting is subject to bias and variance. A model will have a high bias if both train and test errors are high, as shown in Figure 2-13.

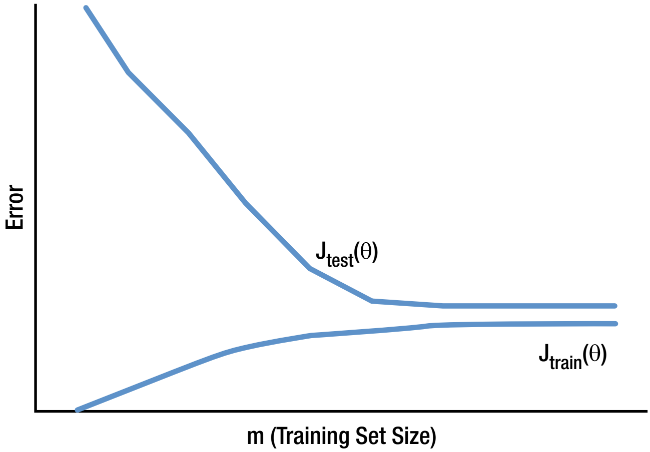


Figure 2-13. *Illustration of high bias*

Moreover, a model will have a high variance if there is a large gap between test and train errors, as shown in Figure 2-14.

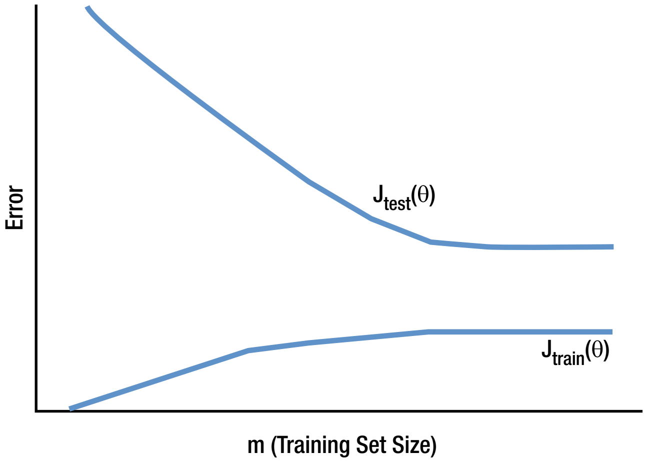


Figure 2-14. *Illustration of high variance*

If a model has a high bias type underfitting, then the remedy can be to increase the model complexity, and if a model is suffering from high variance type underfitting, then the cure can be to bring in more data or otherwise make the model less complex.

Claire decided to randomly divide the data into a train/test split to validate the accuracy of the model and perform cross-validation in cases of overfitting (Listing 2-8).

Listing 2-8. Splitting the Data in Training and Testing sets

```
def split_train_test(data, feature, train_index=0.7):

    train, test = train_test_split(data, test_size = 1-train_index)

    if type(feature) == list:
        x_train = train[feature].as_matrix()
        y_train = train['concrete_strength'].as_matrix()

        x_test = test[feature].as_matrix()
        y_test = test['concrete_strength'].as_matrix()

    else:
        x_train = [[x] for x in list(train[feature])]
        y_train = [[x] for x in list(train['concrete_strength'])]

        x_test = [[x] for x in list(test[feature])]
        y_test = [[x] for x in list(test['concrete_strength'])]

    return x_train, y_train, x_test, y_test
```

Claire explained the code in Listing 2-8 to Smith as follows: "In the code above, we are performing a 0.7-0.3 split. This means that 70% of the observations in the data form the training dataset on which the data is fitted on, while 30% of it comprises the test dataset on which the model will be evaluated."

Claire pointed out that underfitting and overfitting are something that can be detected from the output of a model when first applied. In order to determine the best methodology to eradicate underfitting and overfitting, you have to go through multiple iterations of modeling until you land on a model which is free from both of these. Hence, this is an ongoing exercise and should be done whenever any model (regression, classification, etc.) is applied to the data.

Smith was curious to see whether some techniques exist to evaluate the accuracy of a regression model once applied to a dataset. Claire's answer was affirmative, and hence she compiled the next section for that purpose.

Regression Metrics of Evaluation

Sklearn.metrics is a great tool in determining the performance of a regression model. It does this by implementing several utility functions, scores, and losses. These usually give a single output; however, they have been enhanced to generate multiple outputs as well, most notably, mean absolute error, R^2 score, mean squared error, and explained variance score.

Claire came up with a brief list of the most commonly used evaluation methods for regression.

Explained Variance Score

This score defines the proportion of variance in population explained by the best fit regression model. The best score for this metric is 1. Explained variance score is mathematically defined as follows:

- Formula: $explained_{variance}(y,\hat{y}) = 1 - \frac{Var\{y-\hat{y}\}}{Var\{y\}}$
- Where:
  - $\hat{y}$: Estimated target output
  - y: Corresponding (correct) target output
  - Var: Variance

Mean Absolute Error

Mean absolute error (MAE) is the mean of residuals (i.e., difference between the estimated target and actual target outcomes). Alternative formulations of this measure may include relative frequencies as the weight factors. MAE works on the same scale (i.e., unit) on which the data is measured. Hence the limitation is that there is no possibility of making comparisons between series using different scales/units. This can be mathematically defined by the following:

- Formula: $MAE(y,\hat{y}) = \frac{1}{n_{samples}} \sum_{1=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2$
- Where:
  - $\hat{y}_i$: Estimated target output
  - y_i: Corresponding (correct) target output
  - $n_{samples}$: Number of samples

Mean Squared Error

Mean squared error (MSE) is similar to MAE, except that now the square of residual is taken instead of taking the absolute difference among the estimated target output and corresponding actual target output. The value is always positive and values closer to 0 are better. Taking the square root of MSE will yield root mean square deviation (RMSE) which holds the same units as that of the target output. The mathematical definition of MSE is as follows:

- Formula: $MSE(y,\hat{y}) = \frac{1}{n_{samples}} \sum_{1=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2$

- Where:
  - $\hat{y}_i$: Estimated target output of i[th] sample
  - y_i: Corresponding (correct) target output
  - $n_{samples}$: Number of samples

R²

R², also referred to as the coefficient of determination, defines the measure of how many future samples are likely to be predicted by the regression model. Moreover, it also signifies the proportion of variance in response variable that is predictable from the exploratory ones. Value ranges from 1 to minus infinity where 1 is the most optimum. The mathematical representation is as follows:

- Formula: $R^2\left(y,\hat{y}\right)=1-\frac{\sum_{1=0}^{n_{samples}-1}\left(y_i-\hat{y}_i\right)^2}{\sum_{1=0}^{n_{samples}-1}\left(y_i-\acute{y}\right)^2}$
- Formula: $R^2=\frac{variance of predicted values \hat{y}}{variance of observed values y}$
- Example:
  - *R = -0.7*
  - *R² = 0.49 (i.e., half the variation is accounted for by linear relationship)*

- Where:
  - $\hat{y}_i$: Estimated target output of i[th] sample
  - y_i: Corresponding (correct) target output
  - $\acute{y}=\frac{1}{n_{samples}}\sum_{1=0}^{n_{samples}-1}y_i$
  - $n_{samples}$: Number of samples

Residual

- Formula: ***Residual = observed y – predicted y*** OR $(y-\hat{y})$
- Mean of least squares residuals is always 0

Residual Plot

A residual plot is a scatter plot of regression residuals against exploratory variable x (i.e., independent). See Figure 2-15. The least square regression line is pulled by and toward a point that is extreme in x direction with no other points near it. A dotted line represents the least square regression line.

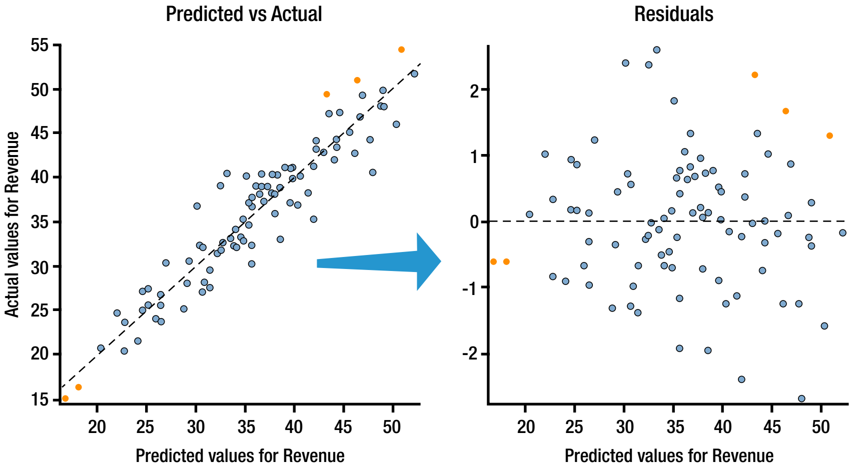


Figure 2-15. *Residual plot*

Residual Sum of Squares

The residual sum of squares (RSS) refers to amount of variance in data not captured by the best fit line of a regression model. The more minimal the RSS, the better the model is.

For the sake of simplicity we will go forward with using R^2 as the metrics of cross-validation. Moreover, it seems to be a better choice considering that we are working for prediction, and this metric will help us determine where future samples will lie on the accuracy scale.

Now having gone through the concepts of underfitting and overfitting and evaluation techniques of regression, Claire believed that it was time for her to test their mantle in finding a regression model which best fits the data at hand.

Types of Regression

Claire continued on by pointing out that SKLearn is the most common library for machine learning in Python. SKLearn offers a spectrum of algorithms for regression modeling. She came up with Figure 2-13 from SKLearn's web site as a reference to determine the regression algorithms which are usually the best given the size of our dataset.

You can download the cheat sheet shown in Figure 2-16 from http://scikit-learn.org/stable/tutorial/machine_learning_map/.

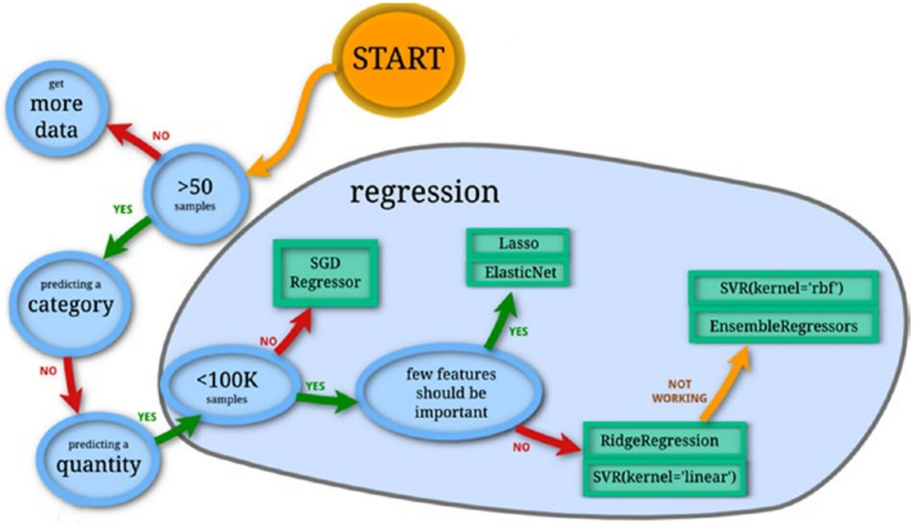


Figure 2-16. *Cheat sheet of Regression-SKLearn*

Smith decided to give it a shot and try to identify the regression models suitable for the size of the current dataset. He recalled having said: "As in our case, the number of samples is less than 100K, hence the potential models are Lasso, ElasticNet, SVR(kernel='rbf'), Ensemble Regressors, Ridge Regression, and SVR(kernel='linear')."

Linear regression is the most basic of the regression models to start with, and hence Claire decided to start modeling from it.

Linear Regression

Smith already knew what a linear regression was, thanks to Claire (see Figure 2-2). Hence Claire decided to put it into the application to see how it performs, so she compiled the code in Listing 2-9.

Listing 2-9. Calculating Single Linear Regression

```
plt.figure(figsize=(15,7))
plot_count = 1

for feature in ['cement_component', 'flay_ash', 'water_component',
'superplasticizer', 'coarse_aggregate']:
    data_tr = data[['concrete_strength', feature]]
    data_tr=data_tr[(data_tr.T != 0).all()]

    x_train, y_train, x_test, y_test = split_train_test(data_tr, feature)
```

```
    # Create linear regression object
    regr = LinearRegression()

    # Train the model using the training sets
    regr.fit(x_train, y_train)
    y_pred = regr.predict(x_test)

    # Plot outputs
    plt.subplot(2,3,plot_count)

    plt.scatter(x_test, y_test,  color='black')
    plt.plot(x_test, y_pred, color='blue',
             linewidth=3)
    plt.xlabel(feature.replace('_',' ').title())
    plt.ylabel('Concrete strength')

    print feature, r2_score(y_test, y_pred)

    plot_count+=1

plt.show()
```

Output

```
cement_component     0.250526602238
flay_ash             0.0546142086558
water_component      0.112291736027
superplasticizer     0.0430808938239
coarse_aggregate     0.0206812124102
```

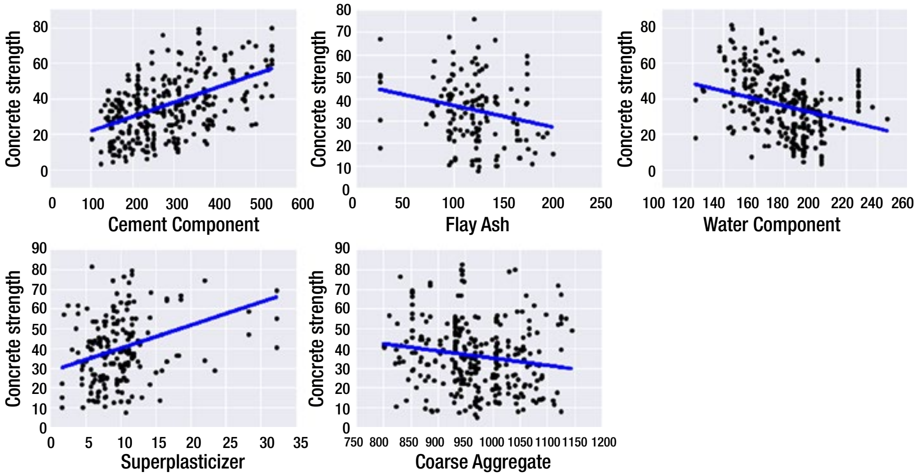


Figure 2-17. *Single linear regression plots*

Explaining the code was essential for Smith to understand the outcome. Thus Claire came up with the following explanation of the code in Listing 2-9:

> *While looking at the code above, you will notice that we are only using the exploratory variables in which we noticed either positive or negative correlations while visually looking at Figure 2-8. Second we have initialized a figure for subplots with a width of 15 and height of 7. Third we did the train/test split method for cross-validation.*

Smith decided to take a shot at explaining what the output means. Almost all of the exploratory variables have their R^2 values closer to 0, which indicates that the linear regression best fit line failed to capture the variance that lay within each of these. However, an exception lies in the case of the cement component, which has a relatively decent variance captured by its best fit line. Considering that the R^2 is low, neither of the models can be used for predicting concrete strength.

Listing 2-10. Calculating Multiple Linear Regression

```
features = ['cement_component', 'flay_ash', 'water_component',
'superplasticizer', 'coarse_aggregate']

data_tr = data
data_tr=data_tr[(data_tr.T != 0).all()]

x_train, y_train, x_test, y_test = split_train_test(data_tr, features)

# Create linear regression object
regr = LinearRegression()

# Train the model using the training sets
regr.fit(x_train, y_train)
y_pred = regr.predict(x_test)

plt.scatter(range(len(y_test)), y_test,  color='black')
plt.plot(y_pred, color='blue', linewidth=3)

print 'Features: %s'%str(features)
print 'R2 score: %f'%r2_score(y_test, y_pred)
print 'Intercept: %f'%regr.intercept_
print 'Coefficients: %s'%str(regr.coef_)
```

Output

```
Features:      ['cement_component', 'flay_ash', 'water_component',
               'superplasticizer', 'coarse_aggregate']
R2 score:      0.097982
Intercept:     77.802791
Coefficients:  [0.04531335  0.01168227 -0.13620573  0.24324622 -0.0329745
```

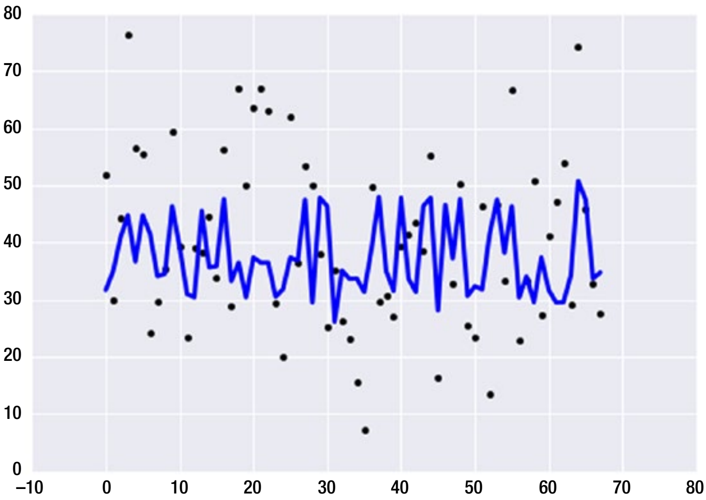


Figure 2-18. *Multiple linear regression plot*

Claire was surprised, as Smith was spot-on. She was curious to see what would happen if she ran a multiple regression by incorporating all of the exploratory variables at the same time. Hence she came up with the code snippet in Listing 2-10. Smith was able to understand the code to a certain extent; however, he was surprised to see a non-linear regression best fit line in Figure 2-18.

His perception was that multiple linear regression should be producing a linear line rather than a non-linear best fit line. To this Claire responded the following:

> *Number of features in analysis is directly proportional to the number of dimensions on the plot. As in our case we have five exploratory and one response variable which add up to six variables. Plotting points in 6 dimensions will be a difficult thing to do, hence the representation of that is shown using two dimensions in Figure 2-18. The best fit line would have been linearly represented had we seen that in the 6 dimensions.*

Claire pointed out that since the R^2 for this multiple regression is extremely low, one cannot use this model to predict the response variable (i.e., concrete strength). Note that the R^2 is extremely low; hence we will definitely not go forward with this to answer the question at hand. However, the output also gives us the coefficients. The first is the alpha coefficient, with the subsequent ones representing the beta coefficients. Both Smith and Claire knew that multicollinearity exists in the data. Hence in the pursuit of finding regression techniques which work on a multicollinear data, they came across Ridge regression and decided to try that out. However, Claire thought it better to explain to Smith about grid search. Before proceeding onward, let's tap into other regression models to find the one that fits our data in the best possible way.

Grid Search

Grid search uses a 'fit' and 'score' methodology to determine the best parameter for a given model. The model to be tuned along with the parameters and their finite possible values is passed within GridSearchCV. The output signifies the parameter values on which the model will be best tuned.

Having explained grid search, Clare compiled a knowledge base regarding Ridge regression.

Ridge Regression

Ridge regression implements the loss function of linear least squares function with a regularization of L2-norm. This type of regression has an inbuilt support of accepting multiple exploratory variables to predict the response variable. L2 is a vector norm which captures the magnitude of a vector. L2 is most commonly taught and used frequently. Consider that we have a vector $\vec{\beta}$ which has two components β_0 and β_1. L2-norm for this can be defined as follows:

- Formula: $\left\|\vec{\beta}\right\|_2 = \sqrt{\beta_0^2 + \beta_1^2}$

This is the Cartesian distance from the origin to these vector components (i.e., β_0 and β_1). Ridge regression is good in the following conditions:

- When we have more than one exploratory variable
- Multicollinearity exists between the exploratory variables

Multicollinearity will lead to biased estimators because the beta coefficients get abnormally high when calculated over many iterations. Hence this will lead us to the question, the best coefficient of which iteration is the real one? As the estimators will be biased now, taking the coefficient averages over much iteration won't yield the population coefficients. However, the variances within these coefficients won't be too high and thus will help in better understanding these coefficients. In order to avoid having beta values that are too big we need to put some constraint on how big the betas can get. Hence we can mathematically define Ridge regression as follows:

- Formula: $\min_{\vec{\beta}} \left\|\vec{y} - A\vec{B}\right\|_2^2$ **subject to** $\vee \vec{\beta} \vee_2^2 \leq c^2$

  OR

  $\min_{\vec{\beta}} \left\|\vec{y} - A\vec{B}\right\|_2^2 + \lambda \vee \vec{\beta} \vee_2^2$

In Ridge regression considering our choice is L2-norm, our beta value constraint is in the shape of a circle which can be seen in Figure 2-19. Earlier we applied single and multiple linear regression on data. However, Ridge, Lasso, and ElasticNet regression are more suitable with multiple regression and thus will be our analyses of choice.

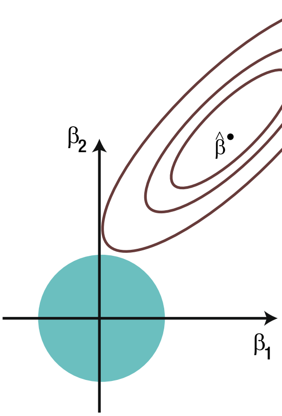


Figure 2-19. *Representation of Ridge regression*

Now that Smith understood Ridge regression, he was interested to see how well it predicts concrete strength from the available exploratory variables. As we read earlier, Ridge regression has a support of accepting multiple exploratory variables as input; hence, Claire decided to run a multiple Ridge regression, as seen in Listing 2-11. Figure 2-20 shows the result.

Listing 2-11. Calculating Multiple Ridge Regression

```
alphas = np.arange(0.1,5,0.1)

model = Ridge()
cv = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas))

y_pred = cv.fit(x_train, y_train).predict(x_test)

plt.scatter(range(len(y_test)), y_test,  color='black')
plt.plot(y_pred, color='blue', linewidth=3)

print 'Features: %s'%str(features)
print 'R2 score: %f'%r2_score(y_test, y_pred)
print 'Intercept: %f'%regr.intercept_
print 'Coefficients: %s'%str(regr.coef_)
```

Output

```
Features:      ['cement_component', 'flay_ash', 'water_component',
               'superplasticizer', 'coarse_aggregate']
R2 score:      0.097982
Intercept:     77.802791
Coefficients:  [0.04531335  0.01168227 -0.13620573  0.24324622 -0.03297459]
```

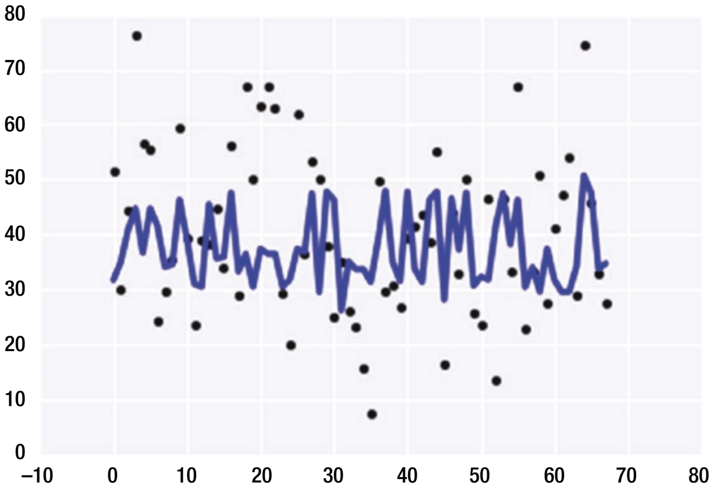


Figure 2-20. *Multiple Ridge regression plot*

Smith didn't have much difficulty in understanding the code in Listing 2-11 as it resembled the code in Listing 2-10. The only thing he didn't understand was how grid search was brought into application. Hence Claire started to explain the piece of code that has to do with GridSearchCV. She at first pointed to alphas = np.arange(0.1,5,0.1) which was used to generate an array of values in the range of 0.1 to 5, with an offset of 0.1. This along with the Ridge regression model was then fed into GridSearchCV. Hence GridSearchCV tried values of alpha one by one to determine the one that best fine-tunes the model.

Having understood the code in Listing 2-11, Smith had a question. Did R^2 fare better when applying Ridge regression? To this Claire said the following:

> *Not at all, as it's exactly the same and so are the coefficients. The reason for which can be that in the presence of multiple features, and multicollinearity linear regression tends to bias towards the L2-norm, and L2-norm is the regularization term which is used by Ridge regression, hence a resemblance. Moreover, also notice that we are using grid search in our approach to optimize for alpha. So what exactly is alpha? Alpha is a regularization parameter which is used to weight the L2-norm term in Ridge regression. The value of 0 for alpha translates the Ridge regression model to the ordinary least squares regression model. Hence the higher the value of alpha, the higher the smoothness constraint, and the lower would be the magnitude of the coefficients. The effect of alpha values is shown for the illustration in Figure 2-21.*

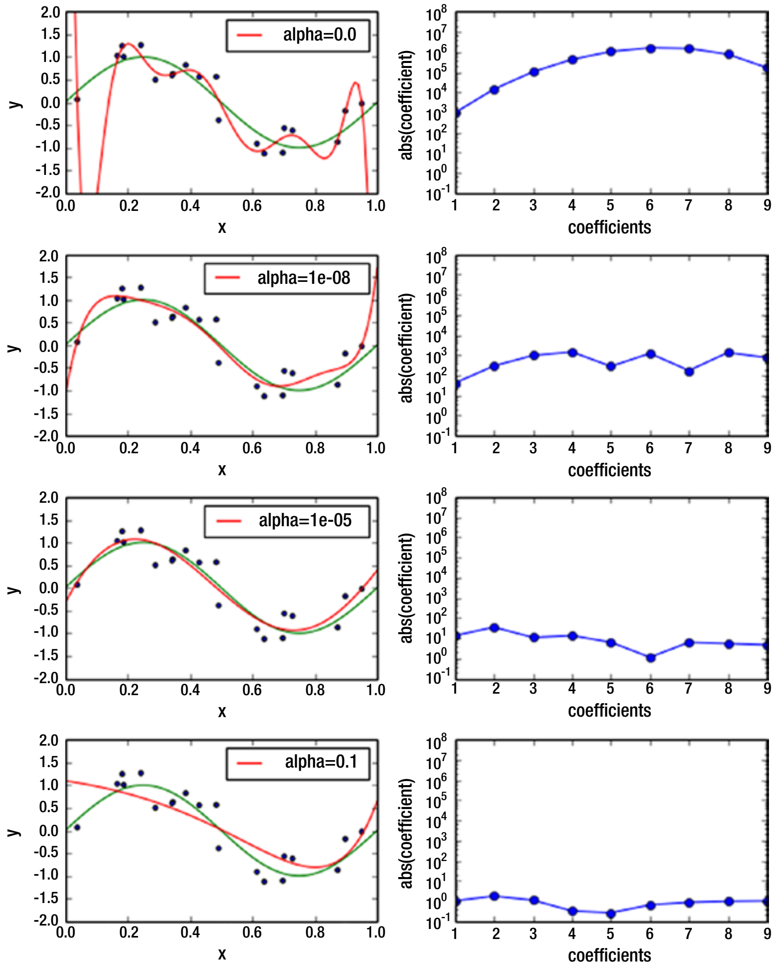


Figure 2-21. *Effect of alpha value on the smoothness of the regression fit line*

Ridge regression even after parameter didn't bring any improvement over what a multiple linear regression had to offer. Claire believed that changing the regularization term from L2-norm to L1-norm might bring some improvements to the R^2 score. Hence, she decided to try out Lasso regression, and came up with the description of that in the following sections.

Lasso Regression

Contrary to Ridge regression, Lasso regression uses an L1-norm. L1-norm for Lasso regression can be defined as follows:

- Formula: $\left\|\vec{\beta}\right\|_1 = |\beta_0| + \beta_1 \vee$

Mathematically we define Lasso regression as follows:

- Formula: $\min_{\vec{\beta}} \left\|\vec{y} - A\vec{B}\right\|_2^2$ **subject to** $\vee \vec{\beta} \vee_1 \leq c$

  OR

- $\min_{\vec{\beta}} \left\|\vec{y} - A\vec{B}\right\|_2^2 + \lambda \vee \vec{\beta} \vee_1$

Because our Ridge regression choice was an L2-norm, our beta value constraint was in the shape of a circle. However, in Lasso regression which takes L1-norm into consideration, our beta constraint is in the shape of a diamond (see Figure 2-22).

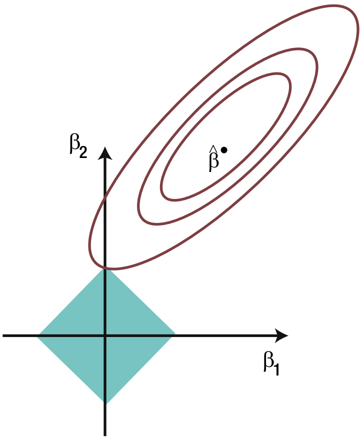


Figure 2-22. *Representation of Lasso regression*

Considering the edge of this diamond lying on the x axis of the first quadrant we can see that the coordinates are (c,0). The level curve above touches the diamond at (0,c) which means that we have a value for β_1 and not for β_2. Hence substituting the following in the equation will lead to the exploratory variable associated with β_1 to turn into 0. Thus Lasso regression in this entire process will end with only a few exploratory variables as compared to Ridge regression in which the shape is a circle and includes all exploratory variables within the dataset.

Smith was anticipating an improvement in the R^2 here because he believed that handpicking only a few exploratory variables and neglecting the rest can make the model stronger. Hence, Claire tested the model, as shown in Listing 2-12.

Listing 2-12. Calculating Multiple Lasso Regression

```
model = Lasso()
cv = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas))

y_pred = cv.fit(x_train, y_train).predict(x_test)

plt.scatter(range(len(y_test)), y_test,  color='black')
plt.plot(y_pred, color='blue', linewidth=3)

print 'Features: %s'%str(features)
print 'R2 score: %f'%r2_score(y_test, y_pred)
print 'Intercept: %f'%regr.intercept_
print 'Coefficients: %s'%str(regr.coef_)
```

Output

```
Features:       ['cement_component', 'flay_ash', 'water_component',
                'superplasticizer', 'coarse_aggregate']
R2 score:       0.103610
Intercept:      77.802791
Coefficients:   [0.04531335 0.01168227 -0.13620573 0.24324622 -0.03297459]
```

Smith noticed that Lasso regression fared better than multiple Ridge and linear regressions (Figure 2-23).

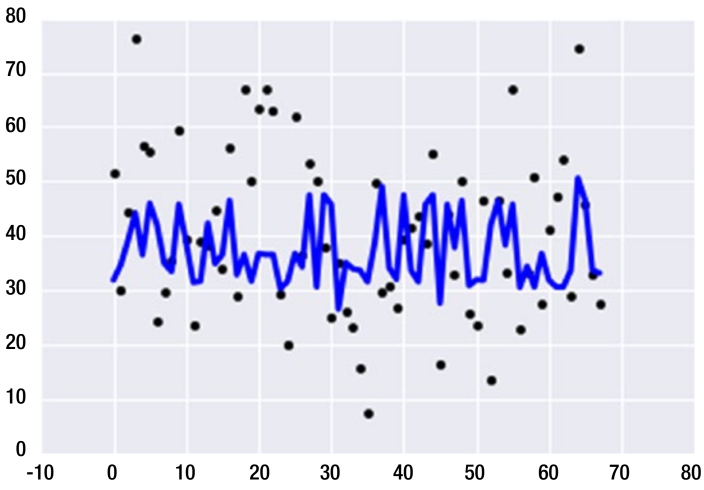


Figure 2-23. *Multiple Lasso regression plot*

However, as per Claire, a R^2 of 0.1 is very low and not something the response variable can be extrapolated onto. Also, she pointed out that grid search was used again to optimize for alpha term as was done in Ridge regression.

Having had no major success so far, Claire decided to bring in the best of the regressions. ElasticNet brings in an amalgam of both of these two approaches (i.e., Ridge and Lasso).

ElasticNet

ElasticNet overcomes the limitations found in Lasso regression (i.e., the penalty function). Lasso tends to select only some exploratory variables and for the multicollinear group of exploratory variables, it will only select one from the group. To avoid this, it is wise to add a quadratic part to the penalty (i.e., $\|\beta\|^2$) which is present in Ridge regression. Hence what ElasticNet does is to include the convex sums of Lasso and Ridge regression penalties.

Claire wrote the code in Listing 2-13 to try multiple ElasticNet on the data.

Listing 2-13. Calculating Multiple ElasticNet Regression

```
model = ElasticNet()
cv = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas))

y_pred = cv.fit(x_train, y_train).predict(x_test)

plt.scatter(range(len(y_test)), y_test,  color='black')
plt.plot(y_pred, color='blue', linewidth=3)

print 'Features: %s'%str(features)
print 'R2 score: %f'%r2_score(y_test, y_pred)
print 'Intercept: %f'%regr.intercept_
print 'Coefficients: %s'%str(regr.coef_)
```

Output

```
Features:      ['cement_component', 'flay_ash', 'water_component',
               'superplasticizer', 'coarse_aggregate']
R2 score:      0.099785
Intercept:     77.802791
Coefficients:  [0.04531335  0.01168227 -0.13620573  0.24324622 -0.03297459]
```

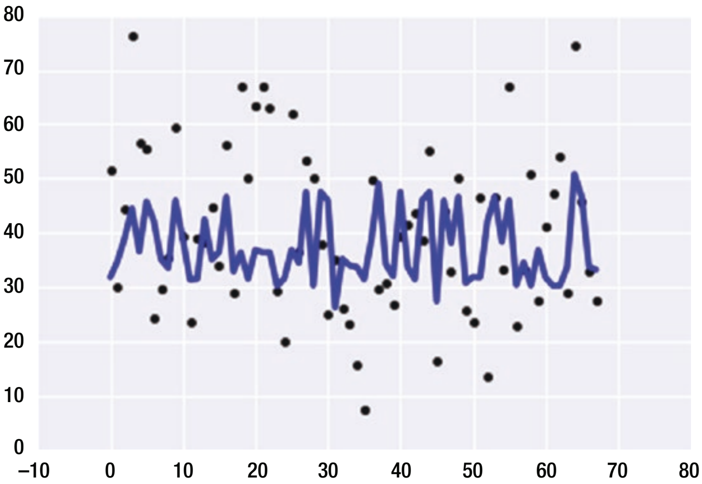


Figure 2-24. *Multiple ElasticNet regression plot*

Claire was disappointed with the results of this model (Figure 2-24). She recalled:

> *The cross-validation seemed to be lower here as compared to Lasso regression. As we read earlier, Lasso seems to consider a chunk of the exploratory variables; hence the ElasticNet's efforts in adding both of them together nullifies the advantage that Lasso had in isolation. This is the behavior we saw on the data at hand but beware it doesn't stand universal for all problem sets.*

Smith was curious to if any technique exists that iteratively improves the accuracy of a regression model. Claire knew the answer to that (i.e., gradient boosting regression).

Gradient Boosting Regression

An ensemble of either classification or regression tree models populates into a gradient boosted model (see Figure 2-25). Boosting is a non-linear flexible regression technique that helps increase the accuracy of trees by assigning more weights to wrong predictions. The reason for inducing more weight is so the model can emphasize more on these wrongly predicted samples and tune itself to increase accuracy. The gradient boosting method solves the inherent problem in boosting trees (i.e., low speed and human interpretability). The algorithm supports parallelism by specifying the number of threads.

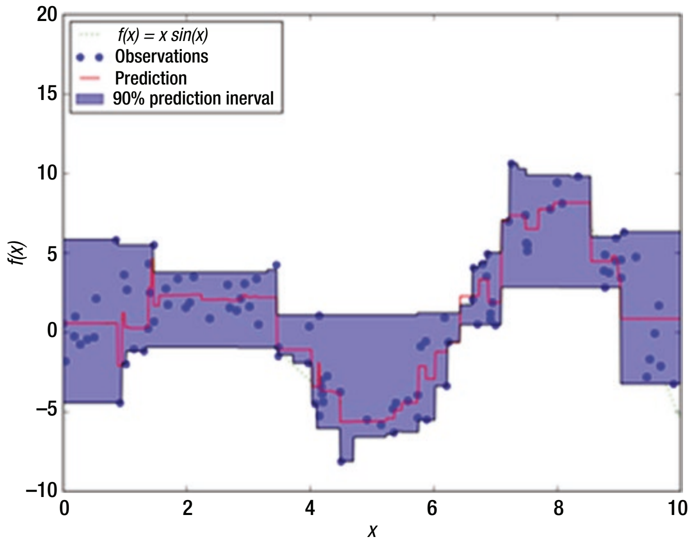


Figure 2-25. *Illustration of gradient boosting regression*

Claire decided to run both single and multiple gradient boosting regressions on the dataset in question. She started out by writing the code for single gradient boosting regression in Listing 2-14.

Listing 2-14. Calculating Single Gradient Boosting Regression

```
plt.figure(figsize=(15,7))
plot_count = 1

for feature in ['cement_component', 'flay_ash', 'water_component',
'superplasticizer', 'coarse_aggregate']:
    data_tr = data[['concrete_strength', feature]]
    data_tr=data_tr[(data_tr.T != 0).all()]

    x_train, y_train, x_test, y_test = split_train_test(data_tr, feature)

    # Create linear regression object
    regr = GradientBoostingRegressor()

    # Train the model using the training sets
    regr.fit(x_train, y_train)
    y_pred = regr.predict(x_test)

    # Plot outputs
    plt.subplot(2,3,plot_count)
```

```
    plt.scatter(x_test, y_test,  color='black')
    plt.plot(x_test, y_pred, color='blue',
             linewidth=3)
    plt.xlabel(feature.replace('_',' ').title())
    plt.ylabel('Concrete strength')

    print feature, r2_score(y_test, y_pred)

    plot_count+=1

plt.show()
```

Output

```
cement_component     0.339838267592
flay_ash             0.0804872388797
water_component      0.311858270879
superplasticizer     0.123130891086
coarse_aggregate     0.230383758064
```

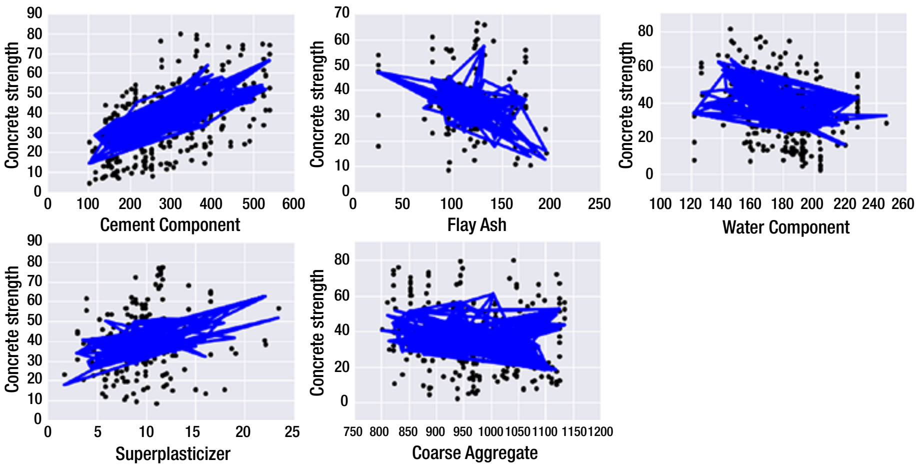


Figure 2-26. *Single gradient boosting regression plot*

Smith was thrilled to see the results as after much effort they were able to significantly improve the R^2 coefficient of the regression model (Figure 2-26).

While looking at the R^2 coefficient values of cement component, water component, and coarse aggregate, Smith saw a better R^2 relative to what they had gotten from linear regression. He knew that the results could be improved by incorporating grid search to tune in the parameter values. Claire was still not content with the results; hence she went forward to run multiple gradient boosting in Listing 2-15.

Listing 2-15. Calculating Multiple Gradient Boosting Regression

```
model = GradientBoostingRegressor()

y_pred = model.fit(x_train, y_train).predict(x_test)

plt.scatter(range(len(y_test)), y_test,  color='black')
plt.plot(y_pred, color='blue',
         linewidth=3)

print 'Features: %s'%str(features)
print 'R2 score: %f'%r2_score(y_test, y_pred)
print 'Intercept: %f'%regr.intercept_
print 'Coefficients: %s'%str(regr.coef_)
```

Output

```
Features:       ['cement_component', 'flay_ash', 'water_component',
                'superplasticizer', 'coarse_aggregate']
R2 score:       0.005876
Intercept:      77.802791
Coefficients:   [0.04531335  0.01168227 -0.13620573  0.24324622 -0.03297459]
```

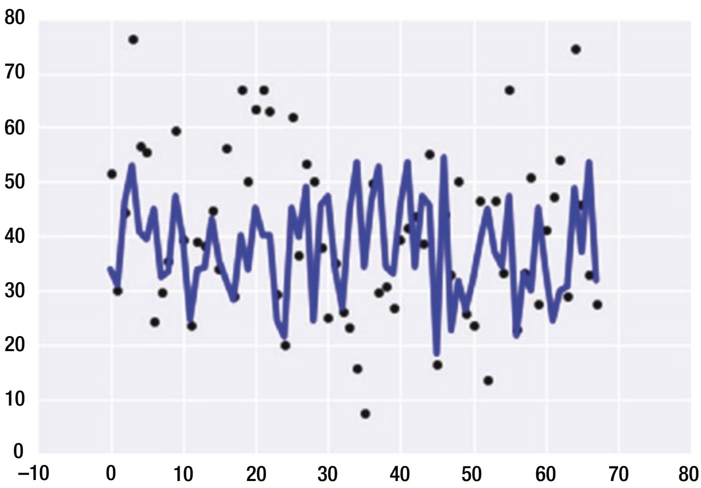


Figure 2-27. *Multiple gradient boosting regression plot*

Smith saw the R^2 go from bad to worse. Single gradient boosting regression seems to have fared better than multiple gradient boosting regression (Figure 2-27).

Claire had once read that to better classify or extrapolate the data, one approach can be to plot the data in a high-dimensional space. Support vector machines tend to do that in the presence of kernels.

Support Vector Machines

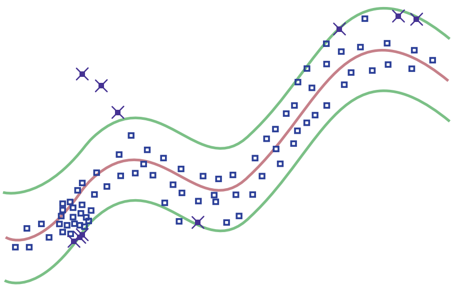


Figure 2-28. *Illustration of support vector machine regression*

Support vector machines constructs hyperplane(s) for the purposes of classification and regression (see Figure 2-28). The goal is to have the largest separation between the two classes. This is ensured by maximizing the distance between the hyperplane and the data point on each side. During this process the sets are often not linearly separable in that space, and thus we are advised to map it to a high-dimensional space (i.e., by introducing kernels).

Claire decided to run both single and multiple support vector machine regressions on the dataset. She also planned to apply the linear kernel to the support vector regressor for better results.

Listing 2-16. Calculating Single Support Vector Machine Regression Using Linear Kernel

```
plt.figure(figsize=(15,7))
plot_count = 1

for feature in ['cement_component', 'flay_ash', 'water_component',
'superplasticizer', 'coarse_aggregate']:
    data_tr = data[['concrete_strength', feature]]
    data_tr=data_tr[(data_tr.T != 0).all()]

    x_train, y_train, x_test, y_test = split_train_test(data_tr, feature)

    # Create linear regression object
    regr = SVR(kernel='linear')

    # Train the model using the training sets
    regr.fit(x_train, y_train)
    y_pred = regr.predict(x_test)

    # Plot outputs
    plt.subplot(2,3,plot_count)
```

```
    plt.scatter(x_test, y_test,  color='black')
    plt.plot(x_test, y_pred, color='blue', linewidth=3)
    plt.xlabel(feature.replace('_',' ').title())
    plt.ylabel('Concrete strength')

    print feature, r2_score(y_test, y_pred)

    plot_count+=1

plt.show()
```

Output

```
cement_component     0.186215229943
flay_ash             0.0566844466086
water_component      0.0824723749594
superplasticizer     0.0412024702221
coarse_aggregate     0.0293294512993
```

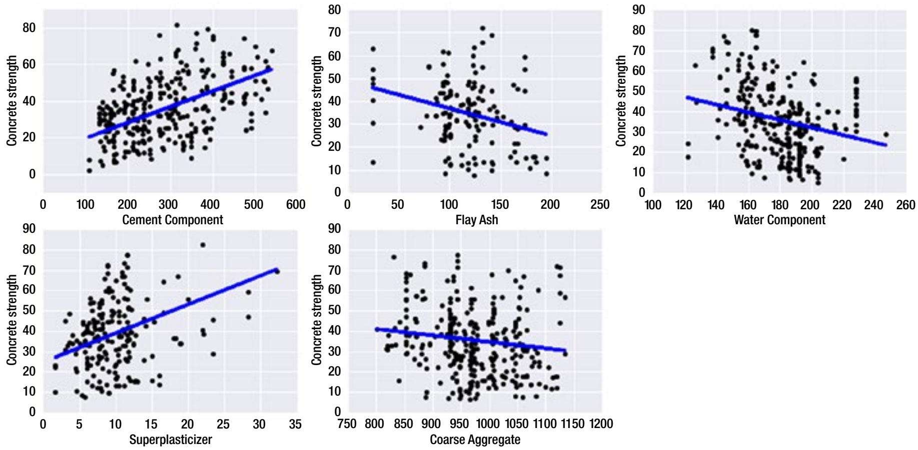


Figure 2-29. *Single support vector machine regression plot*

Smith noticed that contrary to the gradient boosting regressor's best fit lines in Figure 2-23, best fit lines in Figure 2-26 seem to be much cleaner, linear, and relatively less overfitted. However, the R^2 got a hit as it decreased from what they observed in single gradient boosting regressor (see Figure 2-29).

Claire came up with the code in Listing 2-17 to see if multiple support vector machines will bring any improvement to the R^2 coefficient. Figure 2-30 shows the result.

Listing 2-17. Calculating Multiple Support Vector Machine Regression Using Linear Kernel

```
model = SVR(kernel='linear')

y_pred = model.fit(x_train, y_train).predict(x_test)

plt.scatter(range(len(y_test)), y_test,  color='black')
plt.plot(y_pred, color='blue', linewidth=3)

print 'Features: %s'%str(features)
print 'R2 score: %f'%r2_score(y_test, y_pred)
```

Output

```
Features:  ['cement_component', 'flay_ash', 'water_component',
           'superplasticizer', 'coarse_aggregate']
R2 score:  0.010077
```

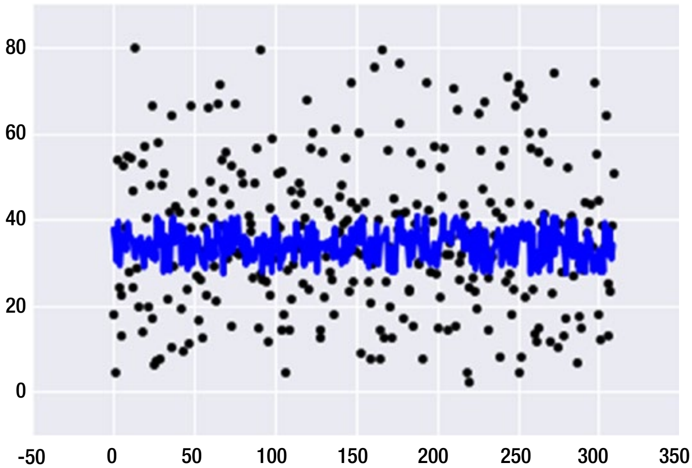


Figure 2-30. *Multiple Support Vector Machine regression plot*

On Smith's scale, the support vector regressor performed the worst among the models they had explored earlier. Claire believed that parameter tuning can be done to improve the performance. Moreover, other kernel types can be put into practice to increase the efficiency of the predictions.

Smith and Claire are done for the day. Help them tune the model, and try different kernels by completing the following exercises.

EXERCISES

1. Run the support vector machine regression with 'rbf' kernel. Also try running 'polynomial' and 'sigmoid' kernels to optimize for prediction accuracy.
2. Gradient boosting regressor has parameters of learning rate, `min_samples_split`, `min_samples_leaf` on which it can be tuned. Run a grid search over these to increase the efficiency of the regressor.
3. Support vector regressor has parameters of c and gamma on which it can be tuned onto. Run a grid search over these to increase the efficiency of the regressor.
4. Gradient boosting regressor returns the feature importance of every model fitted. Explore features which the model gave more importance to, and redo the modeling by only considering those models together or in isolation.
5. In an earlier exercise we split the data on the basis of age and computed correlations. Pick the age split which had the greatest correlation and then repeat the examples and exercises to see if we can fit the model with higher precision for that group of data.

Applications of Regression

Applications of regression are vibrant in several fields of study.

Predicting Sales

Regression can be used to predict the sales of a good or service such as demand analysis (Figure 2-31).

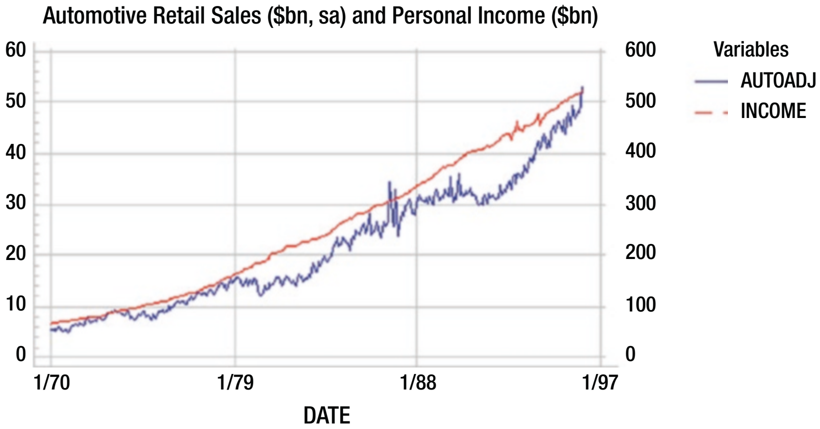


Figure 2-31. *Prediction of retail sales and personal income*

Historical data of sales can be used to extrapolate the results for the future. Features like marketing costs, merchandizing, price, and number of stock keeping units (SKUs) can be given as the exploratory variables to the model.

Predicting Value of Bond

Inflation rate at different times can be used to predict bonds' value. This can be the game changer when estimating the expected return from a portfolio that includes bonds.

Rate of Inflation

In economics we have the concepts of rate of inflation and theory of money supply. If a researcher believes the rate of inflation to be a function of the amount of money supply in the economy then what can he do? He can make rate of inflation as a response variable and the supply of money as the exploratory variable (see Figure 2-32).

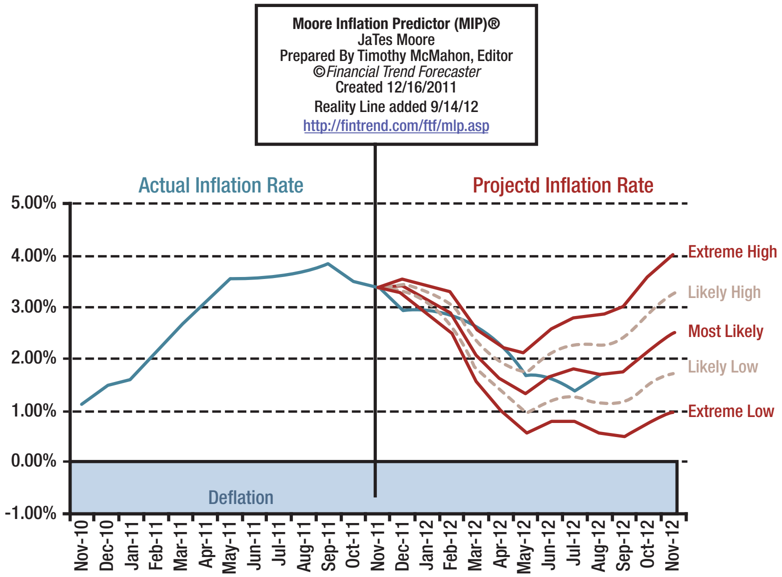


Figure 2-32. *Prediction of inflation rates*

Insurance Companies

These companies use regression to predict the amount of people who might reclaim their health insurance or life insurance, or the number of people who might have accidents in their insured vehicles.

Call Center

A call center manager might want to know the increase in number of complaints as a function of the call waiting times.

Agriculture

Regression can be used to predict the number of fruits that a given region is expected to yield. Possible exploratory variables can be the amount of rainfall, hours of sunshine, number of diseases affecting crops, number of affected crops by fire or diseases, quality index of soil, land fertility index, and so on.

Predicting Salary

Universities can use a predictive model to forecast the salary of students. Possible exploratory variables can be grades, number of competitions participated in, seat placement in the class, number of internships done, number of research papers published, number of projects done, number of sport events participated in, and so on.

Real Estate Industry

The price of real estate can be predicted from size in square feet of a property, number of rooms, availability of parking, square feet of open space, number of miles from main road, and so on (see Figure 2-33).

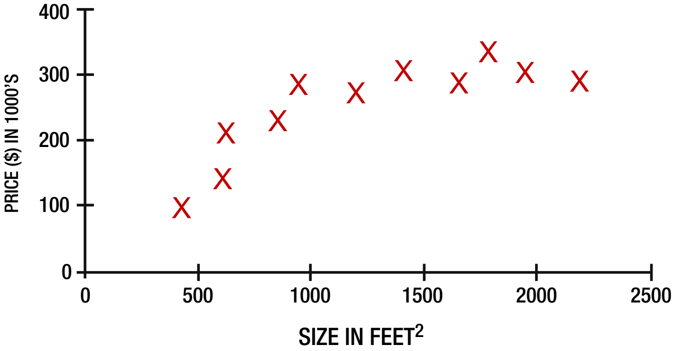


Figure 2-33. *Predicting property as a function of size in feet*

To sum up this chapter, Claire and Smith at first came up with a consensus to use regression for predicting concrete strength. They planned to do so by means of a regression equation in order to avoid the costs of procuring testing equipment to measure it. They started off with the analysis by at first understanding the correlations that exist among concrete strength and exploratory variables. The ones exhibiting a positive or negative correlation were used for regression modeling. Moreover, correlations helped discover the multicollinearity that existed among the three exploratory variables.

Regression model evaluation metrics were looked at and R^2 was picked for evaluating the regression models. They looked at what grid search is and how to apply it to determine the best parameter values for best tuning the model. They split the data into test and train subsets to enable cross-validation. Among the many regression techniques applied, single gradient boosting regressor exhibited the best R^2 values. Hence by the end of the analysis Smith and Claire had the following three models which they could use in isolation to predict concrete strength.

At first Claire defined the use case of the gradient boosting regressor model which takes cement component as an input to predict concrete strength. She recalled from the output of Listing 2-14 that the confidence level of this regression model is approximately 34%.

Listing 2-18. Predicting Concrete Strength from Cement Component

```
feature = 'cement_component'
cc_new_data = [213.5]

data_tr = data[['concrete_strength', feature]]
data_tr=data_tr[(data_tr.T != 0).all()]
```

```
x_train, y_train, x_test, y_test = split_train_test(data_tr, feature)

regr = GradientBoostingRegressor()

# Train the model using the training sets

regr.fit(x_train, y_train)
cs_pred = regr.predict(cc_new_data)
print 'Predicted value of concrete strength: %f'%cs_pred
```

Output

```
Predicted value of concrete strength: 34.008896
```

As an illustration, an input value of cement component was passed in the variable 'cc_new_data.' Claire pointed out that in order to predict concrete strength, a new value can be passed in the same variable.

Then Claire repeated the exercise, the difference being that water component was taken as an input. She recalled from the output of Listing 2-14 that the confidence level of this regression model is approximately 31%.

Listing 2-19. Predicting Concrete Strength from Water Component

```
feature = 'water_component'
wc_new_data = [200]

data_tr = data[['concrete_strength', feature]]
data_tr=data_tr[(data_tr.T != 0).all()]

x_train, y_train, x_test, y_test = split_train_test(data_tr, feature)

regr = GradientBoostingRegressor()

# Train the model using the training sets
regr.fit(x_train, y_train)
cs_pred = regr.predict(wc_new_data)
print 'Predicted value of concrete strength: %f'%cs_pred
```

Output

```
Predicted value of concrete strength: 35.533469
```

As an illustration an input value of water component was passed in the variable 'wc_new_data.'.

Finally, Claire repeated the exercise, the difference this time being that coarse aggregate was taken as an input. She recalled from the output of Listing 2-14 that the confidence level of this regression model is approximately 23%.

Listing 2-20. Predicting Concrete Strength from Coarse Aggregate

```
feature = 'coarse_aggregate'
ca_new_data = [1000]

data_tr = data[['concrete_strength', feature]]
data_tr=data_tr[(data_tr.T != 0).all()]

x_train, y_train, x_test, y_test = split_train_test(data_tr, feature)

regr = GradientBoostingRegressor()

# Train the model using the training sets
regr.fit(x_train, y_train)
cs_pred = regr.predict(ca_new_data)
print 'Predicted value of concrete strength: %f'%cs_pred
```

Output

```
Predicted value of concrete strength: 32.680344
```

As an illustration, an input value of water component was passed in the variable 'ca_new_data'.. Smith was convinced by the approaches taken by Claire, and he believed that if some more time was invested on R&D (research and development), he could pitch this alternative to management and free them of any need to procure any more equipment.

CHAPTER 3

Time Series

The goal of this chapter is to get you started with time series forecasting. A time series forecast is different from regression in that time acts as an exploratory variable and should be continuous along equal intervals. The chapter will cover the concept of stationary, its importance, and methodologies to check its existence in a time series object. Several time series models will be applied, and their forecasts will be checked using the most effective evaluation techniques.

Note This book incorporates **Python 2.7.11** as the de facto standard for coding examples. Moreover, you are required to have it installed for the *Exercises*.

Case Study: Predicting Daily Adjusted Closing Rate of Yahoo

David had a meeting scheduled with his client Janice the next day. They were meeting to investigate the reason for a dip in Janice's returns. Janice was the daughter of a millionaire businessman, and she had approached David after hearing speculations that the market was expected to grow with escalating economic conditions. She was interested in taking a long-term risky position to enjoy potentially high returns on investment. However, this quarter she had seen a dip in her returns despite a good outlook in the capital market.

David searched Janice's portfolio and started looking at the performance of each stock individually to see how each one of them had fared in the last quarter. He figured out that the reason for the mismatch between the portfolio's return and the expected returns was a huge residual in Yahoo stock forecasts. This forecast error would have had a minimal impact had the number of shares been a tiny chunk of the entire portfolio, but that was not the case in this scenario.

David recalled the courses in corporate finance and business analytics that he had taken during his time at Harvard Business School and decided to give those concepts a try for figuring out a solution to this problem. Should he apply regression or time series? Which techniques can he apply to a forecast for the Yahoo stock?

David pulled the data regarding Yahoo stock from the *Matplotlib finance* package provided by Python (see Listing 3-2 for a code snippet).

D. Haroon, *Python Machine Learning Case Studies*, DOI 10.1007/978-1-4842-2823-4_3

David did some research to see what information is present in each feature within the data. He came up with a data dictionary for the *Yahoo Stock prices* dataset (see Table 3-1).

Table 3-1. *Data Dictionary for the Yahoo Stock Prices Dataset*

Feature name	Description
Date	Date of given day of trading
Year	Year of given day of trading
Month	Month of given day of trading
Day	Month of given day of trading
D	Floating point representation of day
Open	Open rate of given day of trading
Close	Close rate of given day of trading
High	High rate of given day of trading
Low	Low rate of given day of trading
Volume	Shares volume of given day of trading
adjusted_close	Close rate of given day of trading. This also takes into account distributions and corporate actions that occurred before that day's market opening.

Before moving forward David thought of initializing the following packages. This he preferred to do in order to avoid bottlenecks while implementing the code snippets on his local machine.

Listing 3-1. Importing Packages Required for This Chapter

```
%matplotlib inline

import pandas as pd
import numpy as np
from datetime import datetime
import statsmodels.api as sm
import matplotlib.pyplot as plt
from IPython.display import Image
from matplotlib.pylab import rcParams
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
from sklearn.linear_model import LinearRegression
from statsmodels.tsa.arima_model import ARMA, ARIMA
from sklearn.metrics import explained_variance_score
from matplotlib.finance import fetch_historical_yahoo, parse_yahoo_
historical_ochl

rcParams['figure.figsize'] = 15, 5
```

David knew with certainty that understanding the data is fundamental prior to starting any analysis. His plan was to load the dataset within the memory and see a time plot of the adjusted closing rates to see if any pattern exists within the time series object.

Feature Exploration

David started off by loading the data into memory.

Listing 3-2. Reading the Data in the Memory

```
fh = fetch_historical_yahoo('^GSPC', (2016, 1, 1), (2016, 12, 31))
yahoo_data = parse_yahoo_historical_ochl(fh, asobject=True, adjusted=False)
```

While explaining the code, he first explained that `fetch_historical_yahoo` class was used to fetch the object of the yahoo index for all trading days of 2016. Next he passed that object into the `parse_yahoo_historical_ochl` method to get the data for the features described in Table 3-1.

Listing 3-3. Plotting Adjusted Closing Rate of Yahoo Stock for 2016 on a Time Series

```
dates = [x[0] for x in list(yahoo_data)]
values = [x[-1] for x in list(yahoo_data)]
data = pd.DataFrame({'Close_Adj':values}, index=dates)

date_thresh = datetime.strptime('2016-10-01',"%Y-%m-%d").date()

data_train = data[:date_thresh]
data_test = data[date_thresh:]

plt.plot(data_train.index, data['Close_Adj'])
plt.xlabel('Year')
plt.ylabel('Adjusted Closing Rate')
plt.title('Yahoo Adjusted closing rate - Year 2016')
```

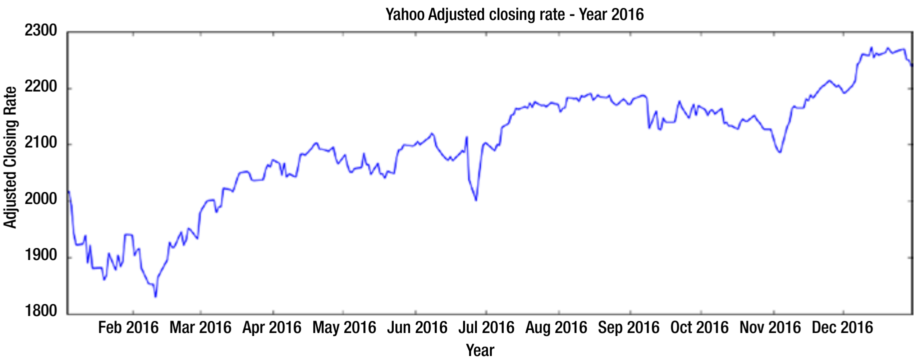


Figure 3-1. *Time series plot of adjusted closing rate of Yahoo stock for year 2016*

From Listing 3-3, David plotted the Yahoo stock closing rates for the year 2016 in Figure 3-1. He split the original data series into test data (i.e., `data_test`) and train data (i.e., `data_train`). Hence, from that time onward David decided to use `data_train` as the data for formulating the time series model.

Looking at the distribution in Figure 3-1 David noticed that a definitive upward trend exists. Moreover, he saw that a seasonality exists (i.e., downward dip), repeating itself every 4.5-month interval.

Because the majority of Janice's portfolio included Yahoo stock, David knew that he had to take some measures to reduce the forecast error. In order for him to validate his model he decided to split the dataset. He aimed to use the data of the first nine months of 2016 as the training dataset, with the last three months for cross-validation of the model's forecast. An idea came mind that he could use the forecasting techniques to come up with a generalization curve for the Yahoo stock time series. Then on the basis of the shape of that curve he could decide on the regression model that would enable predictions at a future point in time. The rationale behind using the forecasting techniques was to capture the seasonality and trend components present in the time series object.

Hence David started off by concentrating on the first goal—that is, formulating a time series forecasting model that captures the components of Yahoo stock time series at its best. He recalled from his corporate finance course that they used to apply time series methods for stock forecasting. He did some research to understand what time series modeling is, and when it should be preferred.

Time Series Modeling

Data points collected at fixed period time intervals constitute a time series. Time series analysis is usually done with the objective of forecasting the long-term trend over time as per the problem's underlying hypothesis.

It was Friday and David was now looking forward to the weekend with his friends and family. He was also excited about meeting his friend Maria, who had been his batch mate in school, whose research was inclined toward statistics.

It was Saturday, and David was having lunch with Maria at a famous fast-food restaurant. Without wasting much time, David brought up the problem and how he was planning on solving it. He knew that it was better to do due diligence in the analysis initially to conclude fruitful research. Thus he was interested to hear from Maria regarding the statistical concepts and techniques essential to time series analysis that he should incorporate before moving forward, Maria recommended that he check if his time series data was stationary as a majority of the time series models are designed to give results on stationary data.

Evaluating the Stationary Nature of a Time Series Object

David broke his plan into pieces. First he planned to see what properties a time series object should have for it to be stationary. Then he had to check if the data at hand was stationary; if it was not, he had to search for the techniques to make it stationary. David started off by pulling up the properties that make a time series object stationary.

Properties of a Time Series Which Is Stationary in Nature

Most of the time series models work on the assumption that data is stationary. Moreover, contrary theories related to stationary time series are easier to implement than non-stationary time series theories. A time series object is stationary if it has the following properties:

- No trend exists
- Mean remains constant over time
- Variance remains constant over time
- No autocorrelation exists. Autocorrelation is the correlation between the series at current time with a lagged version of itself.

Once David knew the four properties a time series object should have for it to be stationary, he started to look for techniques to test if the time series data is stationary or not.

Tests to Determine If a Time Series Is Stationary

His search provided him with two methods for validation: exploratory data analysis and Dickey-Fuller test. The exploratory data analysis method can help detect the property of stationary within the Yahoo stock dataset.

Exploratory Data Analysis

This is more of a visual method of finding if the distribution is stationary. It encompasses the use of rolling mean and rolling variance to answer the underlying question. If these rolling metrics stay constant over time then the distribution will qualify as a stationary time series object. So, how is a rolling mean calculated? First we define the length of a subset over which rolling means have to be calculated (e.g., 5). Then we generate our own time series by averaging over the numbers in series in chunks of 5. Take, for example, a time series with the following values:

10, 50, 34, -5, 15, 19, 1, -30, 16, 37

Then the subsequent rolling mean will be as follows:

NA, NA, NA, NA, NA, 20.8, 22.6, 12.8, 0, 4.2, 8.6

The same holds true for rolling variance except that instead of mean we will be calculating the variance at each instance.

These rolling means will then be plotted on a time series plot. However, by looking at the rolling means above we can deduce that the values seem to vary a lot and are not at all constant. Hence, the time series shown in the example above is not stationary.

Despite the fact that the first method seemed straightforward and easy to implement, David was looking for a statistical technique that would enable him to validate if the time series object is stationary or not. His search also brought him to the Dickey-Fuller test.

Dickey-Fuller Test

This is more of a statistical approach of checking if the time series object is stationary or not. The test in its null hypothesis assumes the time series object to be non-stationary. Once applied to a given time series object, the Dickey-Fuller test returns the test statistics and critical values at different confidence intervals. If the value of test statistics is lower than that of the critical value, then the null hypothesis does not hold true and we opt for the alternative hypothesis (i.e., that the time series object is stationary).

Once he knew what properties make a time series object stationary, and after looking through the techniques to identify it, David decided to check if the data at hand was stationary or not. Thus he came up with the code snippet in Listing 3-4.

Listing 3-4. Method to Evaluate If a Time Series Object Is Stationary

```
def evaluate_stationarity(timeseries, t=30):

    #Determing rolling statistics
    rolmean = timeseries.rolling(window=t).mean()
    rolstd = timeseries.rolling(window=t).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print 'Results of Dickey-Fuller Test:'
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic',
    'p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print dfoutput
```

Writing the code snippet in Listing 3-4 was not an easy task for David, who didn't have considerable experience in programming. He knew that this functionality of time series validation is subject to being reused again and again. Hence he wrote the code in a function so that he could use this functionality whenever needed. He decided to document whatever code he had written so that he could refer to it later, when he faced difficulty in understanding it. As he recalled:

> *By default I initialized the window size 't' to 30, that is, a month. Then I calculated the rolling mean and rolling standard deviation of our time series object. Next I plotted the time series, rolling mean, and rolling*

standard deviation together to apply the exploratory data analysis method into practice. This I knew would enable me to determine if the time series object is stationary or not. Later I decided to also try the statistical test for detection and brought the Dickey-Fuller test into application. I made sure to print the output of the Dickey-Fuller test and determine the critical intervals from the confidence interval.

Now that David had defined the method in Listing 3-4, he decided to call it and pass the time series object as a parameter while keeping the window size 't' to 15 (i.e., two weeks). He called the `evaluate_stationary` method as shown in Listing 3-5.

Listing 3-5. Using the Method to Test If Time Series Object Is Stationary

```
evaluate_stationarity(data_train['Close_Adj'], 15)
```

Output

```
Results of Dickey-Fuller Test:

Test Statistic                -1.114302
p-value                        0.709335
#Lags Used                     0.000000
Number of Observations Used  188.000000
Critical Value (5%)           -2.877040
Critical Value (1%)           -3.465620
Critical Value (10%)          -2.575032
dtype: float64
```

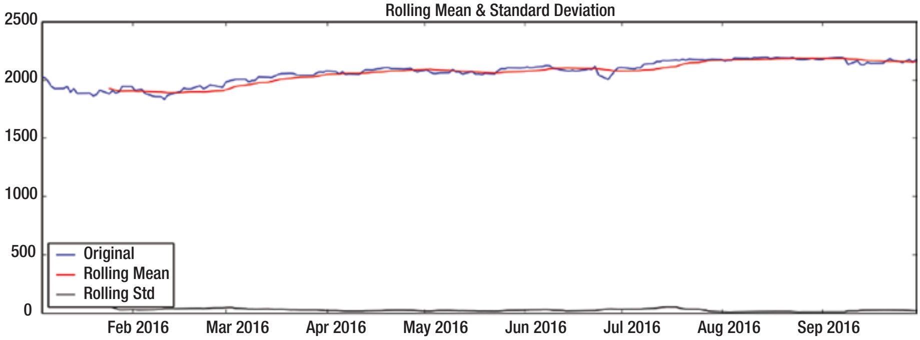


Figure 3-2. *Plot of rolling mean and standard deviation of time series object*

While looking at the exploratory data analysis in Figure 3-2, David noticed that despite the variation in rolling standard deviation being small, rolling mean clearly seems to relatively vary a lot. Hence he deduced the time series object to be non-stationary. He then looked at the output of the Dickey-Fuller test to see if it depicts the same result as well. He noticed the test statistic to be greater than the critical values; hence he failed to reject the null hypothesis, which meant that the data is non-stationary in nature.

The time series object being non-stationary meant that now the path to forecasting was no longer straightforward. He would first have to make the time series object stationary before he could apply forecasting to it. Hence he did some research and came up with several methods to make a time series object stationary.

Methods of Making a Time Series Object Stationary

Though several methods of doing the job exist, none of them promises to make the time series object completely stationary. What these methods do is to transform the time series object to make it look closer to a stationary object. The methods do this by removing trend and seasonality from the time series object. Following are some of the methods that can make a time series object closer to a stationary object:

- Applying transformations
- Estimating trend and removing it from the original series
- Differencing
- Decomposition

David decided to apply all these methodologies one by one to determine the one that can make the time series object as stationary as possible. He started off by looking at how applying transformations can help achieve this short-term objective.

Applying Transformations

Transformations like log, cube root, and square root penalize larger values and thus can enable the series to become stationary. David decided to apply each of these transformations to figure out the one that makes this time series object almost stationary. He started off with log transformation

Log Transformation

David wrote the code snippet in Listing 3-6, to see if log transformation can yield a time series which is stationary in nature.

Listing 3-6. Applying Log Transformation to Time Series Object

```
data_log = np.log(data_train['Close_Adj'])
evaluate_stationarity(data_log, 15)
```

Output

```
Results of Dickey-Fuller Test:

Test Statistic                 -1.136109
p-value                        0.700460
#Lags Used                     0.000000
Number of Observations Used    188.000000
Critical Value (5%)            -2.877040
Critical Value (1%)            -3.465620
Critical Value (10%)           -2.575032
dtype: float64
```

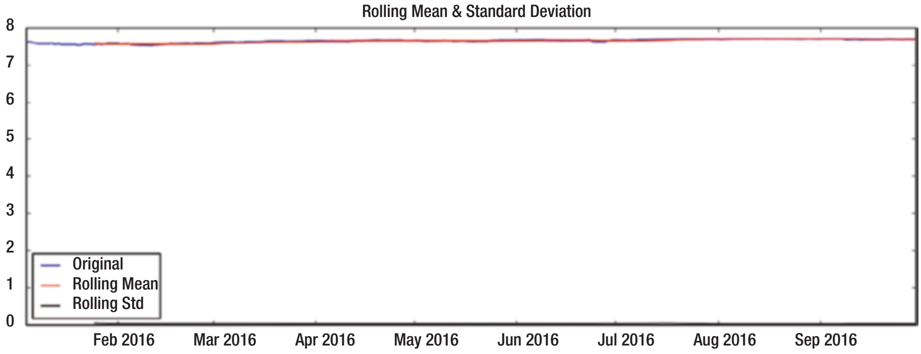


Figure 3-3. *Log transformed time series along with rolling mean and standard deviation*

Looking at Figure 3-3, David noticed that the transformation did work as it managed to make the rolling mean and rolling standard deviation constant. However, he was astonished to see that the Dickey-Fuller test proved otherwise (i.e., the transformed series is still non-stationary). He deduced this after noticing the test statistics to be greater than the critical values which meant that the null hypothesis holds true and the log transformed object is non-stationary in nature. He was curious to know why both methods produced results contradicting each other. The only explanation he could think of was that the rolling mean, though it seems constant, might vary once he zoomed into Figure 3-3. Hence he added some lines of code as shown in Listing 3-7.

Listing 3-7. Zooomed Figure of Rolling Mean from a Log Transformed Distribution

```
data_log = np.log(data_train['Close_Adj'])

#Determing rolling statistics
rolmean = data_log.rolling(window=15).mean()
rolstd = data_log.rolling(window=15).std()
```

```
#Plot rolling statistics:
orig = plt.plot(data_log, color='blue',label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.ylim([7.4,7.8])
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```

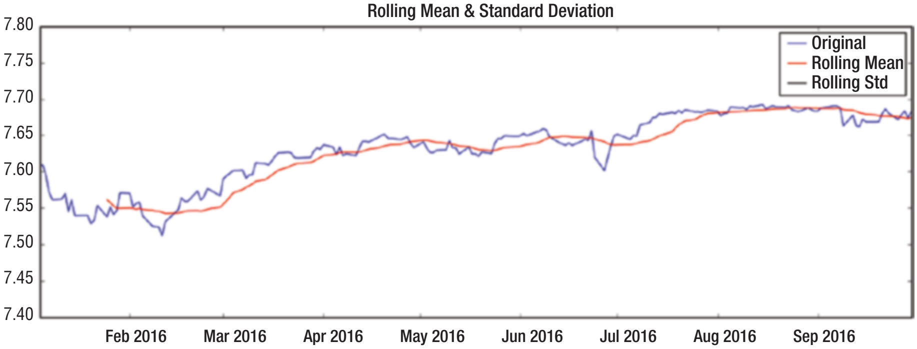


Figure 3-4. *Plot of a zoomed figure of rolling mean from a log transformed distribution*

The zoomed figure in Figure 3-4 revealed rolling mean to be varying and not at all constant against time. According to David, this reasoning showed that the results from the Dickey-Fuller test demonstrated that the transformed series is non-stationary. Log transformation failed to make the object stationary in nature, and thus David moved to square root transformation as the next resort.

Square Root Transformation

David applied square root (Sqrt) transformation to Yahoo stock data by writing the script in Listing 3-8.

Listing 3-8. Applying Sqrt Transformation to Time Series Object

```
data_sqrt = np.sqrt(data_train['Close_Adj'])
evaluate_stationarity(data_sqrt, 15)
```

Output

```
Results of Dickey-Fuller Test:

Test Statistic                  -1.124704
p-value                          0.705121
#Lags Used                       0.000000
```

```
Number of Observations Used     188.000000
Critical Value (5%)             -2.877040
Critical Value (1%)             -3.465620
Critical Value (10%)            -2.575032
dtype: float64
```

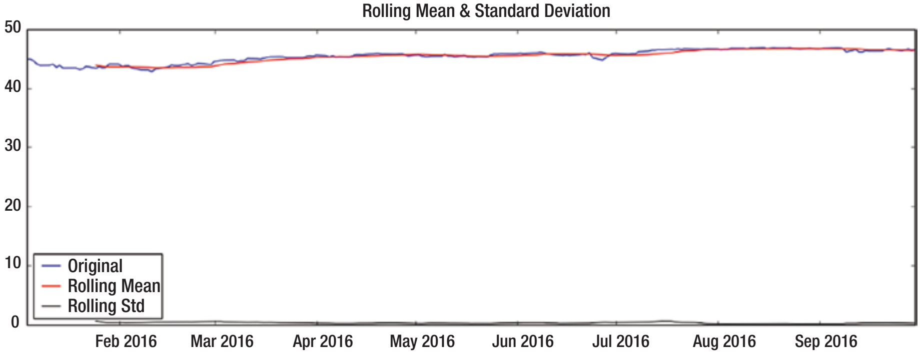


Figure 3-5. *Sqrt transformed time series along with rolling mean and standard deviation*

David deduced the following from Figure 3-5:

> *Rolling mean and standard deviation seem constant as we see the plot above, but not much as compared to what we saw in the log transformed series plot. This is the reason that in the results of this Dickey-Fuller test, test statistics is greater than the critical values, which means that the null hypothesis holds true and the log transformed object is non-stationary in nature.*

Log and square transformation weren't able to make the time series object at hand closer to a stationary nature. Log transformation did help to scale down the time series subject and make it more flat. Hence, David decided to apply other methods to the log transformed series to make it stationary. He started looking for other transformations recommended by the statistical community and that is when he learned about cube transformation. However, it was time for him to leave for home and he thus requested you to apply the cube transformation in your spare time and share the results with him.

EXERCISE

1. Determine the cube transformation of our time series. Does it improve the value of the test statistic?

It was a new day and David resumed with his short-term mission of transforming the nature of the time series object to a stationary one. Having failed with the transformation methodologies, he was in search of other techniques that could help him achieve his short-term objective. He started afresh and started looking at the initial plot of the time series object in Figure 3-1. It was then that an idea came to mind: what if he removed the upward trend from the data to make it stationary.

Estimating Trend and Removing It from the Original Series

He knew that the approach was two-fold: at first he would need to estimate the trend and then remove it from the Yahoo stock time series object. He started looking for techniques for estimating trends and luckily he found two methods: moving average smoothing and exponentially weighted moving average. He started off with moving average smoothing.

Moving Average Smoothing

Moving average smoothing is similar to rolling mean in that in both these methods, the average of observations lying within a fixed window size is calculated in order to estimate the observation at a given time.

David decided to apply moving average smoothing to estimate the trend of the log transformed time series. He decided to apply rolling mean over the past two weeks (i.e., last 15 values) by writing the code in Listing 3-9.

Listing 3-9. Applying Moving Average Smoothing to the Time Series Object

```
moving_avg = pd.rolling_mean(data_log,15)
plt.plot(data_log)
plt.plot(moving_avg, color='red')
```

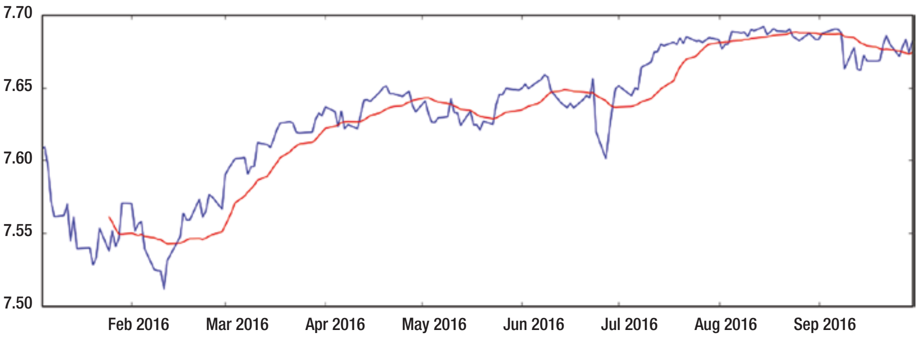


Figure 3-6. *Moving average smoothed time series*

David made the following deduction from Figure 3-6: "The red line shows the rolling mean, that is, the estimated trend from the moving average smoothing. The goal is now to subtract this estimated trend from the log transformed time series." Note that since David had taken the average of the last 15 values, rolling mean is not defined for the first 14 values. This can be observed from the output of Listing 3-10.

Listing 3-10. Printing Trendless Time Series Object

```
data_log_moving_avg_diff = data_log - moving_avg
data_log_moving_avg_diff.head(15)
```

Output

```
2016-01-04       NaN
2016-01-05       NaN
2016-01-06       NaN
2016-01-07       NaN
2016-01-08       NaN
2016-01-11       NaN
2016-01-12       NaN
2016-01-13       NaN
2016-01-14       NaN
2016-01-15       NaN
2016-01-19       NaN
2016-01-20       NaN
2016-01-21       NaN
2016-01-22       NaN
2016-01-25       -0.023439

Name: Close_Adj, dtype: float64
```

David knew that the trend has `NaN` in its first 14 observations. Hence he decided to remove `NaN` from the trend and then subtract this trend from the original time series object. He expected the result to be stationary and planned to test it (see Listing 3-11) from the function defined in Listing 3-4.

Listing 3-11. Evaluating Trendless Time Series for Stationary

```
data_log_moving_avg_diff.dropna(inplace=True)
evaluate_stationarity(data_log_moving_avg_diff)
```

Output

```
Results of Dickey-Fuller Test:

Test Statistic                -4.375719
p-value                       0.000328
#Lags Used                    0.000000
Number of Observations Used   174.000000
```

```
Critical Value (5%)           -2.878298
Critical Value (1%)           -3.468502
Critical Value (10%)          -2.575704
dtype: float64
```

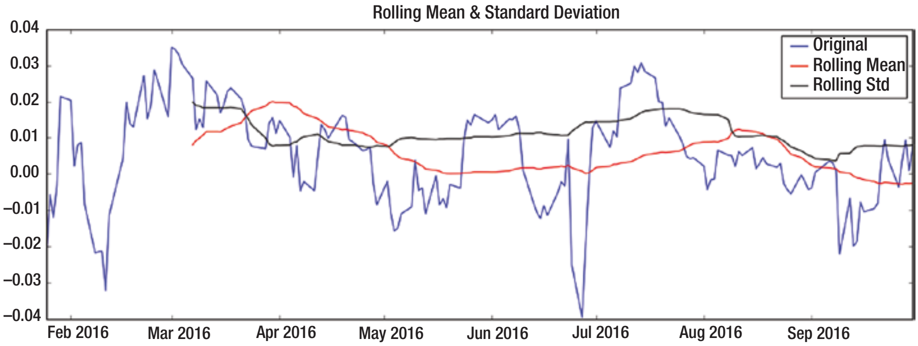


Figure 3-7. *Time series with moving average removed along with rolling mean and standard deviation*

David was thrilled by some concrete results while looking at Figure 3-7. The exploratory analysis approach proved the trendless data to be stationary because of a near to constant rolling mean and standard deviation. The Dickey-Fuller test proved the same as well because the test statistics came out to be smaller than all of the critical values. This meant that the null hypothesis is rejected and the trendless time series object stood stationary with a confidence level of 99%.

David got what he wanted; that is, he changed the nature of the log transformed time series object to a stationary one. However, he wanted to explore if other techniques can make it closer to the stationary nature. Hence, he decided to try the exponentially weighted moving average for estimating the trend.

Exponentially Weighted Moving Average

Exponentially weighted moving average is similar to the rolling mean approach except that a recent observation is assigned the highest weight and a distinct one the lowest. In other words, the weight assignment decreases exponentially from the most recent observation to the most distinct one.

David then applied the exponential weighted moving average to the log transformed time series object to see if he could witness a considerable improvement in making the time series object resemble the stationary one.

Listing 3-12. Applying Exponential Weighted Moving Average Smoothing to the Time Series Object

```
expwighted_avg = pd.ewma(data_log, halflife=15)
data_log_ewma_diff = data_log - expwighted_avg
evaluate_stationarity(data_log_ewma_diff)
```

Output

```
Results of Dickey-Fuller Test:

Test Statistic                 -3.273472
p-value                         0.016107
#Lags Used                      0.000000
Number of Observations Used   188.000000
Critical Value (5%)            -2.877040
Critical Value (1%)            -3.465620
Critical Value (10%)           -2.575032
dtype: float64
```

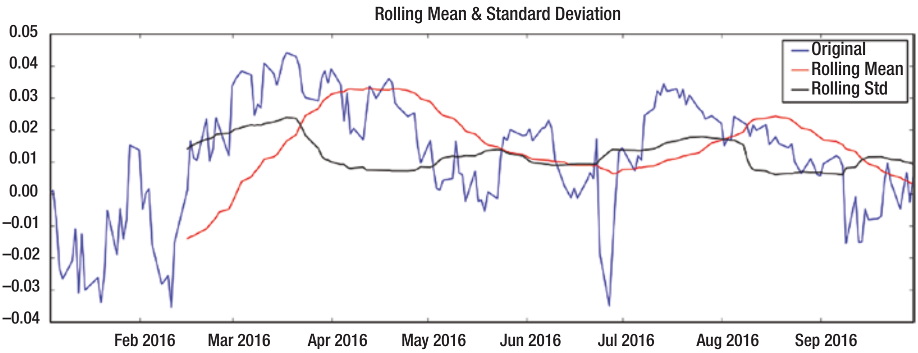


Figure 3-8. *Exponentially weighted moving average smoothed time series*

While looking at Figure 3-8, David noticed that the rolling mean varied more than it did in Figure 3-7. Hence he was he was indifferent to reaching a conclusion using the exploratory data analysis method. However, test statistics from the Dickey-Fuller test barely crossed the 5% critical value measure. Thus, the test statistic being less than the critical value meant rejection of the null hypothesis. This led David to conclude that the log transformed trendless time series object has successfully become stationary in nature.

The only considerable success that David had had so far came from removing the moving average smoothing estimated trend from the log transformed time series. However, he was not convinced of the stationary nature within the time series while looking at its exploratory data analysis plot. As per his deduction in Figure 3-1, the time series object had both trend and seasonality. If trend reduction could do so well, he wondered, logically, whether both trend and seasonality reduction should do even better. He started looking for techniques to reduce both trend and seasonality. It didn't take him long to figure out that differencing can do this job.

Differencing

Differencing subtracts the time series from a lagged version of itself (i.e., observations at the previous instant). Differencing stabilizes the mean of a time series by removing changes in the level. David was confident that differencing, given its definition, would be the more effective in making the time series object stationary.

He was expecting a closer-to-stationary time series subject. Hence he wrote the script in Listing 3-13 for his expectations to come true.

Listing 3-13. Applying First-Order Differencing to the Log Version of the Time Series Object

```
data_log_diff = data_log - data_log.shift()
data_log_diff.dropna(inplace=True)
evaluate_stationarity(data_log_diff)
```

Output

```
Results of Dickey-Fuller Test:

Test Statistic                  -1.038969e+01
p-value                          2.036243e-18
#Lags Used                       1.000000e+00
Number of Observations Used      1.860000e+02
Critical Value (5%)             -2.877208e+00
Critical Value (1%)             -3.466005e+00
Critical Value (10%)            -2.575122e+00
dtype: float64
```

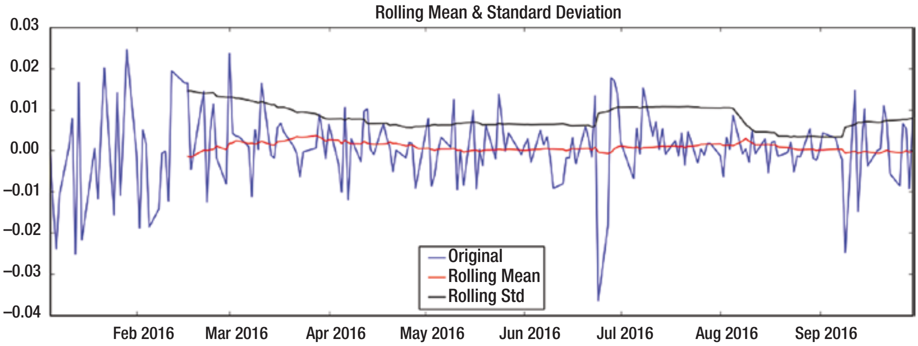


Figure 3-9. *Statistics for the differencing applied time series*

David saw a remarkable improvement in Figure 3-9 as the rolling mean and rolling standard deviation appear to be almost constant. The same was reflected by the Dickey-Fuller test as the test statistics came out to be much less than the 1% critical values. The seasonality and trend removed time series seems to be almost stationary in nature

now. David had applied a first-order differencing; that is, a lag of 1 was induced while differencing. He was curious to find if a second- or third-order differencing can improve the stationary nature of the time series object. He invites you to collaborate on this by attempting the exercises.

EXERCISE

1. Apply second- and third-order differencing to our log transformed series. Did it make the log transformed series stationary?

David's curious mind made him ponder if he could manage to remove seasonality and trend from the log transformed time series object as he had in trend estimation and removal earlier. Earlier he had applied two techniques to estimate the trend and removed that from the time series object to make it stationary in nature. He started looking for methodologies, if any, by which he could break the time series object into trend and seasonality components. He then planned to remove these two components from the time series object to make its resemblance closer to one of a stationary nature. His search brought him to the concept of decomposition.

Decomposition

Decomposition is yet another approach to eliminate trend and seasonality from a time series to make it stationary in nature. It does that by dividing the time series into three components: trend, seasonality, and residuals. The component of interest in this case is the residuals (i.e., time series without trend and seasonality).

David was curious to see what the three components of the log transformed time series in question looked like. Hence, without wasting any further time he wrote the code in Listing 3-14.

Listing 3-14. Decomposing the Log Transformed Time Series

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(list(data_log), freq=15)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(data_log, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
```

```
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```

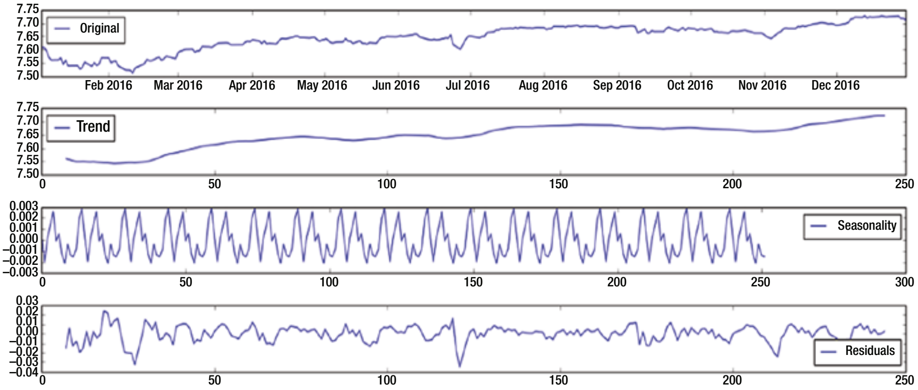


Figure 3-10. *Decomposed log transformed time series*

David had read that trends are linear in nature and the trend component in Figure 3-10 was showing a true depiction of that. However, when David compared this trend component to the trend estimates in Figures 3-6 and 3-8, he noticed that although the estimates depicted the trend, seasonality existed within them as well. The seasonality component in Figure 3-10 resonated with the definition of seasonality as well—that is, a pattern that repeats itself after a fixed time interval. Finally, David deduced the residual component to be fairly constant over time, as it seems in Figure 3-10 as well. Now his aim was to evaluate the residuals to see if they were stationary in nature or not.

Listing 3-15. Evaluating the Residuals for Stationary

```
data_log_decompose = pd.Series(residual)
data_log_decompose.dropna(inplace=True)
evaluate_stationarity(data_log_decompose)
```

Output

```
Results of Dickey-Fuller Test:

Test Statistic                 -7.557158e+00
p-value                         3.077807e-11
#Lags Used                      4.000000e+00
Number of Observations Used     1.700000e+02
Critical Value (5%)            -2.878696e+00
```

```
Critical Value (1%)          -3.469413e+00
Critical Value (10%)         -2.575917e+00
dtype: float64
```

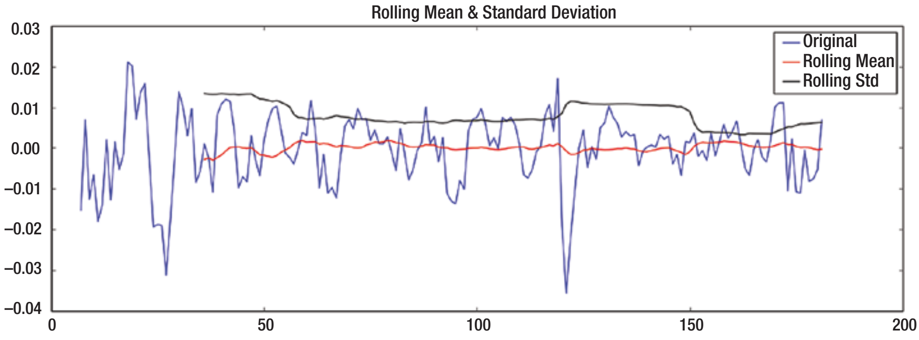


Figure 3-11. *Stationary-related statistics for residuals*

Exploratory data analysis of rolling mean and rolling standard deviation depicted both these measures to be almost stationary over time. David moved forward to see if the Dickey-Fuller test came up with the same verdict. Test statistics came out to be lower than all three critical values. Hence, with a confidence level of 99% he concluded the residual plot to be stationary. However, the test statistics were not much less than the critical values if compared to what we saw in the first-order differencing technique (i.e., the output of Listing 3-13).

David was relaxed to an extent because of the four conditions of a stationary object, he was able to nail the first three by means of the first-order differencing technique. He was able to remove the trend and make the rolling statistics constant over time. Now he had to check for the fourth condition—that is, if autocorrelation exists or not within the time series object. If autocorrelation varies over time for the time series object in question, he would have to figure out a technique to make the autocorrelations constant.

Tests to Determine If a Time Series Has Autocorrelation

Autocorrelation is the correlation between the series at current time with a lagged version of itself. While reading the literature about autocorrelation, David learned that autocorrelation exists in two variants.

Autocorrelation Function

Autocorrelation function (ACF) determines the correlation of time series along with a lagged version of itself. For example, data with ten observations and with a lag of 5 will have the correlation between the following series calculated:

t4, t5, t6, t7, t8, t9

t0, t1, t2, t3, t4, t5

Partial Autocorrelation Function

Partial autocorrelation function (PACF) also measures the correlation of a time series along with a lagged version of itself except that it eliminates the variations already explained by the prior comparisons. For example, as in our case, with a lag of 5, it will remove the variations already captured between t0 and t3.

Now that David knew what autocorrelation is, his next step was to determine the methodologies available to measure it within the log transformed differencing applied time series.

Measuring Autocorrelation

David achieved some success as he was able to find a statistical method to measure autocorrelation within the log transformed time series (i.e., Durbin Watson statistic).

Durbin Watson Statistic

Contrary to the autocorrelation correlogram, Durbin Watson statistic is more of a statistical approach to determining the existence of autocorrelation within the data. It does so by computing autocorrelation among residuals from a statistical regression analysis. It returns a number between 0 and 4 where 0 depicts strong positive autocorrelation, 4 depicts strong negative autocorrelation, and 2 depicts no autocorrelation at all.

While doing the research on autocorrelations, David recalled reading that differencing can be used to eradicate ACF and PACF. Having already applied differencing on the log transformed time series object, David was expecting no presence of ACF and PACF. At first he planned to check for the presence of autocorrelation using the Durbin Watson statistics method in Listing 3-16.

Listing 3-16. Calculating the Durbin Watson Statistics for Log Transformed Differencing Applied Time Series

```
sm.stats.durbin_watson(data_log_diff)
```

Output

```
2.1859293629518555
```

David was thrilled to see the results to be a score approaching 2, which indicated the presence of no autocorrelation in the log transformed differencing applied time series. However, he was curious to see what the ACF and PACF would look in the absence of autocorrelation. Hence he came up with the code snippet in Listing 3-17.

Listing 3-17. Plotting Correlograms for ACF and PACF on Log Transformed Differencing Applied Time Series

```
ax1 = plt.subplot(211)
fig = sm.graphics.tsa.plot_acf(data_log_diff.squeeze(), lags=40, ax=ax1)
```

```
ax2 = plt.subplot(212)
fig = sm.graphics.tsa.plot_pacf(data_log_diff, lags=40, ax=ax2)
```

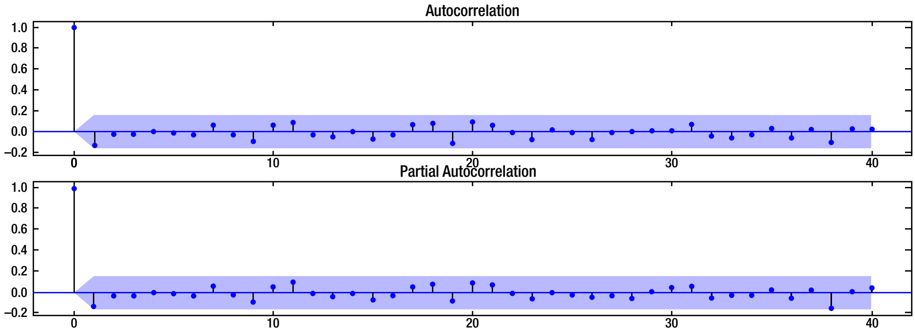


Figure 3-12. *Plotting correlograms for ACF and PACF on log transformed differencing applied time series*

David noticed that both the ACF and PACF correlograms should a spike at 1 lag with no correlations to come on the later time lags. The next step after making the time series stationary was what excited David the most (i.e., applying forecasting to the Yahoo stock dataset to get forecasts with minimal residuals).

Modeling a Time Series

Before applying the different time series models, David thought it better to first understand the underlying concepts many of the models use. After research, he compiled explanations on the meaning of Number of Auto-Regressive (AR), Number of Moving Average (MA), and Number of Differences (D). He deemed these terms important because they formulate the prediction equation for the time series model.

- **Number of Auto-Regressive (AR) terms or p**: AR terms are the lags of the response variable. Take, for example, a value of $p = 5$. Hence, in order to predict the response variable at t5, a time series between t0 and t4 will be considered the exploratory variables.
- **Number of Moving Average (MA) terms or q:** MA terms are the lagged forecast errors in a response variable. Take, for example, a value of q = 5. Hence in order to predict the response variable at t5, a time series between t0 and t4 will be considered. Hence this means the difference between the moving average and actual value at that time instant.
- **Number of Differences (d):** This refers to the order of differencing that we are interested in applying to our time series object.

Parameters for a given time series model are written as follows: p, d, q. Once he understood the common parameters of the time series techniques, David decided to compile a list of the statistical tests to measure the accuracy of the forecasted time series.

Tests to Validate Forecasted Series

After due research, David learned about some of the most notable tests for measuring the accuracy of the forecasted time series.

Mean Forecast Error

This is the mean of residuals at each time point. Mean forecast error values can range from negative infinity to positive infinity.

Mean Absolute Error

Mean absolute error is same as mean forecast error except that the residuals are converted to absolute terms; that is, positive residuals remain the same whereas negative residuals are converted into positive ones. Values can range between zero and positive infinity.

Residual Sum of Squares

The residual sum of squares (RSS) is usually used in regression to understand the fraction of variance not explained by the regression model. It is calculated by calculating the residuals at each point in time, taking their squares, and then adding them over. Squaring of the residuals ensures that the values come out to be positive. RSS can have a minimum value of 0, which indicates that the variance is estimated to the fullest extent by the time series model.

Root Mean Squared Error

The root mean squared error (RMSE), also known as root mean squared deviation (RMSD), is an advanced version of RSS in which square root is applied to the output of RSS.

David now had a fair idea of what was meant by the terms p, q, and d. However, he wasn't sure that what value each of these terms should entail when passing them within a time series model. His search uncovered some of the rules that decide the values of p, q, and d subject to the patterns within ACF and PACF.

Deciding Upon the Parameters for Modeling

David planned to view the ACF and PACF of the log transformed differencing applied time series to determine the rule it best matches.

- **Rule 1**: p = 1 when ACF shows exponential decay, and PACF has a spike at lag 1 with no correlation further on (See Figure 3-13).

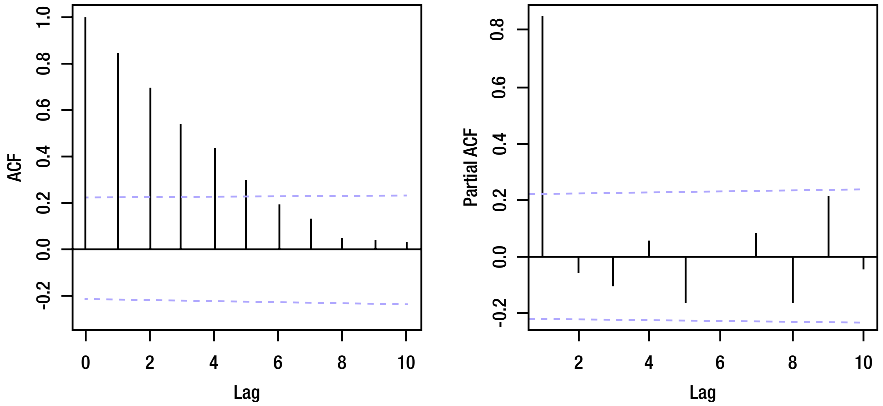


Figure 3-13. *ACF and PACF plots for p = 1*

- **Rule 2**: p = 2, when ACF shows a set of exponential decays or sine wave-shaped pattern, and PACF has spikes at lags 1 and 2, with no correlation further on (See Figure 3-14).

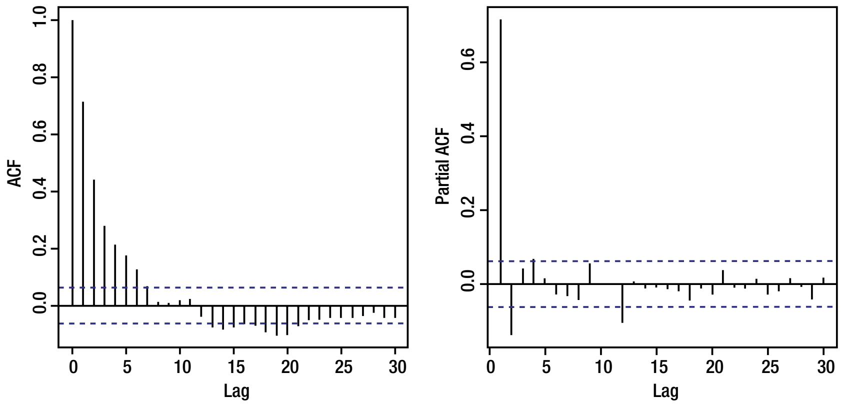


Figure 3-14. *ACF and PACF plots for p=2*

- **Rule 3**: q = 1 when ACF has a spike at lag 1 with no correlation further on, and PACF shows exponential decay (See Figure 3-15).

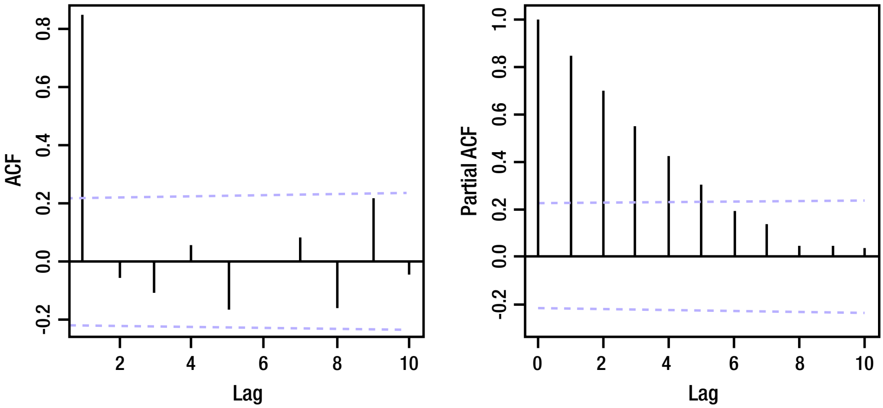


Figure 3-15. *ACF and PACF plots for q = 1*

- **Rule 4**: q = 2, when ACF has spikes at lags 1 and 2, with no correlation further on, and PACF shows a set of exponential decays or sine wave-shaped pattern (See Figure 3-16).

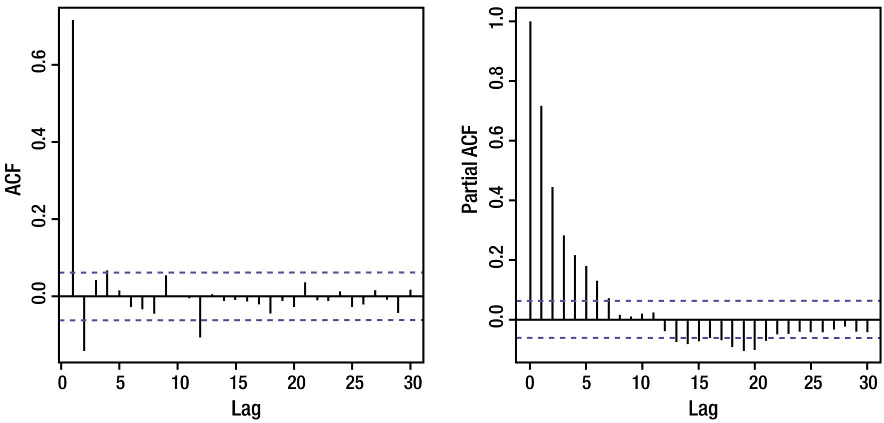


Figure 3-16. *ACF and PACF plots for q = 2*

- **Rule 5**: p = 1, q = 1 when both ACF and PACF show exponential decay starting at a lag of 1 (See Figure 3-17).

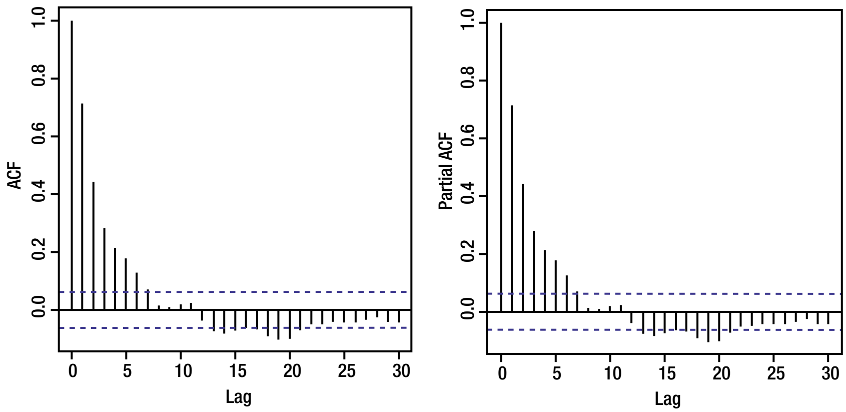


Figure 3-17. *ACF and PACF plots for p = 1 and q = 1*

Looking at Figure 3-12, David wasn't sure that which of the rules satisfied the ACF and PACF patterns obvious in the correlograms. Both ACF and PACF had a spike at 1 with no correlation in the proceeding lags. He didn't find this behavior being covered by any of the rules, and thus he decided to leave to accuracy the values of p, d, and q he would choose for the models. Having gone through the evaluation methods for time series and having gone through the rules for selecting the parameter values, David knew that the time had come to apply the time series models to the differencing applied log transformed time series. His search of time series models brought him to the effectiveness that ARIMA models displayed in various time series applications. Hence he started off with understanding ARIMA.

Auto-Regressive Integrated Moving Averages

ARIMA stands for auto-regressive integrated moving averages. ARIMA forecasting revolves around a linear equation whose behavior is dependent upon the values of p, d, and q. In other words, the ARIMA model filters signal from noise and forecasts it for future point in time. An ARIMA model is defined as ARIMA (p, d, q).

This section discusses some of the variants of an ARIMA model.

Auto-Regressive Moving Averages

The ARMA (auto-regressive moving averages) model is similar to an ARIMA model except that now we use p and q to determine the linear equation. Moreover, now the prediction is done on a mean adjusted series; that is, a time series is transformed to a zero mean variant by subtracting the sample mean from it.

This phenomenon can be defined by the following equation:

$$y_t = Y_t - \acute{Y},\ t = 1, \ldots N$$

Where,

- Y_t : Original time series
- $\acute{Y}$: Mean adjusted time series

ARMA model is defined as ARMA (p, q)

Auto-Regressive

An ARMA model forecasts a time series from a linear function of its past values. Time lag is defined in the AR (auto-regressive) model to forecast the future values of a time series. An AR model can be defined by the following equation:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \epsilon_t$$

Where,

- β_0, β_1 : Beta coefficients
- y_{t-1}: Lagged version of the time series
- ϵ_t : Residuals, which are assumed to be normally distributed and random in time

An AR model is defined as AR (p, d)

David decided to apply the AR model on differencing applied log transformed time series. He knew that because this technique does not have the term 'q,' its value will be 0 (i.e., q = 0). He decided to apply AR with the arbitrary term of p = 2 and d = 1 in Listing 3-18.

Listing 3-18. Applying AR Model to Log Transformed Differencing Applied Time Series

```
model = ARIMA(data_log, order=(2, 1, 0), dates=data_log.index.to_datetime())
results_AR_210 = model.fit(disp=-1)
plt.plot(data_log_diff)
plt.plot(results_AR_210.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_AR_210.fittedvalues-data_log_diff)**2))
```

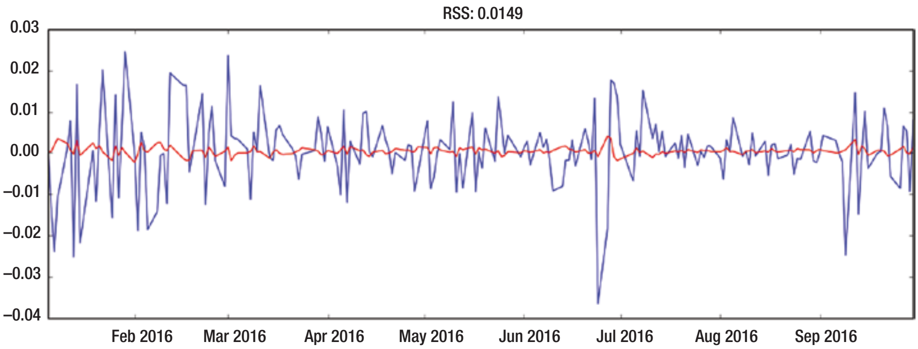


Figure 3-18. *Plotting MA model smoothing on log transformed differencing applied time series*

David made sure to plot the original differencing applied and log transformed time series along with the forecasted one using the AR technique in Figure 3-18. He was pleased to see the RSS for this forecasted series approaching 0, which meant that the model was fairly accurate in capturing the variance in the input series. The prediction of Yahoo stock at that much of a negligible RSS was a big relief for him. He then moved on to search for other techniques and that is when he discovered the moving average technique.

Moving Average

MA stands for moving average model. MA is best for situations in which we have a univariate time series object. MA forecasts future values by training itself on the current and past values of a time series that are random in nature. The MA model is defined as MA (d, q).

MA models don't have the term p; hence David initialized it as p = 0. David decided to apply the MA model to the log transformed differencing applied time series, with a q = 1 and d = 1.

Listing 3-19. Applying MA Model to Log Transformed Differencing Applied Time Series

```
model = ARIMA(data_log, order=(0, 1, 1), dates=data_log.index.to_datetime())
results_MA_011 = model.fit(disp=-1)
plt.plot(data_log_diff)
plt.plot(results_MA_011.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_MA_011.fittedvalues-data_log_diff)**2))
```

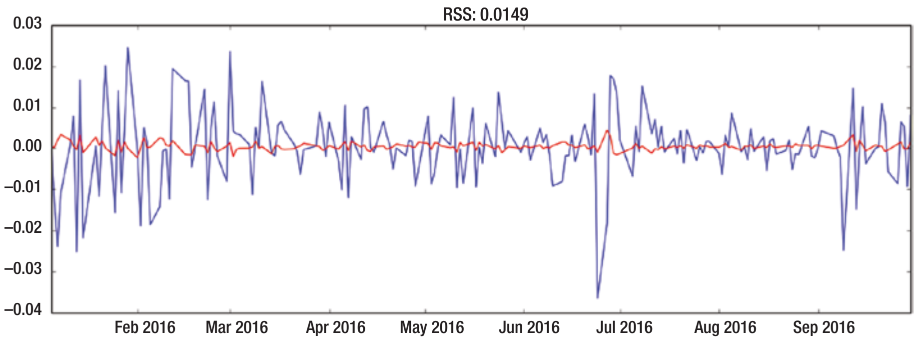


Figure 3-19. *Plotting MA model smoothing on log transformed differencing applied time series*

David noticed no significant improvement in the RSS score after applying MA to the differencing applied log transformed time series. He decided to merge the AR and MA models in the hopes of seeing a more robust model for predicting Yahoo stock.

Combined Model

The combined model refers to the addition of AR and MA. This model will have all the three parameters—p, d, and q—and will initially be defined as an ARIMA —ARIMA (p, d, q). A combined model means that all of the parameters will be taken into consideration. David was anticipating a reduction in the RSS score; hence, without wasting any undue time he applied the combined model in Listing 3-20 with parameter values of p = 1, d = 1, and q = 1.

Listing 3-20. Applying Combined Model to Log Transformed Differencing Applied Time Series

```
model = ARIMA(data_log, order=(1, 1, 1), dates=data_log.index.to_datetime())
results_ARIMA_111 = model.fit(disp=-1)
plt.plot(data_log_diff)
plt.plot(results_ARIMA_111.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_ARIMA_111.fittedvalues-
data_log_diff)**2))
```

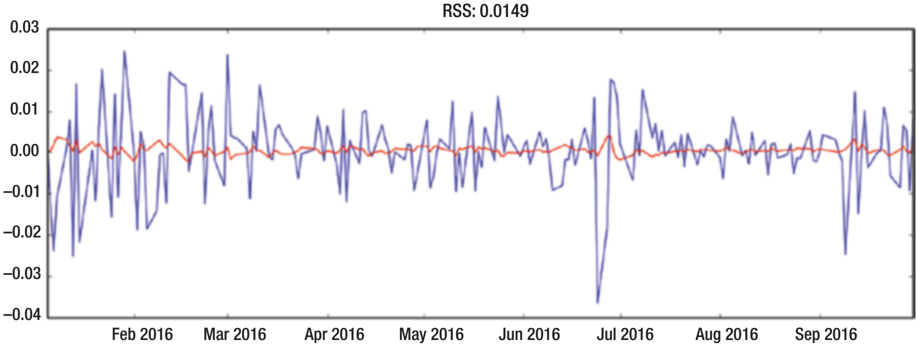


Figure 3-20. *Plotting combined model smoothing on log transformed differencing applied time series*

RSS for this forecasted series was approaching 0 as well; hence the model is fairly accurate in capturing the variance in the input series. However, David was puzzled to see that the RSS scores for all of these approaches were the same. Making a random pick, he decided to go forward with the combined model.

It has been a long journey so far. David applied log transformation, used differencing to make the time series stationary, and finally applied the combined model to that log transformed differencing applied time series. The time series was now different from what he had seen initially because he had applied the log transformation and differencing. As the forecasting was performed on the modified time series, so it now had to be scaled back to see how well it forecasted the initial Yahoo stock time series.

Scaling Back the Forecast

Before scaling back the forecasts to the original units, David thought it wise to print initial observations of the forecasted time series.

Listing 3-21. Printing First Few Observations of the Forecasted Series

```
predictions_ARIMA_diff = pd.Series(results_ARIMA_111.fittedvalues,
copy=True)
print predictions_ARIMA_diff.head()
```

Output

```
2016-01-05    0.000401
2016-01-06    0.000237
2016-01-07    0.001781
2016-01-08    0.003630
2016-01-11    0.003035
dtype: float64
```

The output of Listing 3-21 provided an interesting insight for David. Data loaded initially in Listing 3-2 had dates starting from 2016-01-04 whereas dates in the output of Listing 3-2 were starting from 2016-01-05. After much thinking he found the answer to that. 'The parameter d'—that is, the number of differences—had a value of 1 when forecasting was done using the combined modeling. This induced a lag of 1, and the thus the first element didn't have anything in the lagged version of itself to subtract from. While conducting research on time series models, David had also read that the output of these models gives absolute changes rather than cumulative sums at each time interval. Going forward with absolute changes would have been a sound strategy had the data been discrete. However, in a situation like this, where the forecasted series will be a continuous distribution, cumulative sums were the requirement. Hence David decided to convert these absolute changes to cumulative sums for all time intervals in Listing 3-22.

Listing 3-22. Printing Cumulative Sum of the Forecasted Series

```
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
print predictions_ARIMA_diff_cumsum.head()
```

Output

```
2016-01-05    0.000401
2016-01-06    0.000638
2016-01-07    0.002418
2016-01-08    0.006049
2016-01-11    0.009084
dtype: float64
```

David knew that the model performed well in minimizing the residuals as seen from the RSS score. However, he was also concerned with the quality of the model, and hence he started looking for methods of measuring it. His search concluded when he found Akaike Information Criterion (AIC) to test the quality of a time series model. As per the explanation, the lower the score of AIC, the better the quality of a time series model. Hence he put the forecasted time series to the test in Listing 3-23.

Listing 3-23. Printing AIC BIC, and HQIC scores of the Forecasted Series

```
print results_ARIMA_111.aic
```

Output

```
(-1234.0564146247877)
```

The score from the output of Listing 3-23 came out to be extremely low, thus proving the goodness of the model. Now David's plan of action was to scale back the forecasted series by reversing the differencing and log transformation. He aimed to do so by taking the following steps:

- Remove differencing from the forecasted series.
- Reverse the log transformation by applying an exponent.
- Evaluate the forecasted series by calculating mean forecast error, mean absolute error, and root mean absolute error.
- Plot the original and forecasted series.

David started off by concentrating on removing differencing from the forecasted series. He planned to do that by adding a base value to all observations of the forecasted series. He assumed the first value of the log transformed time series to be the base value and went forward with the implementation in Listing 3-24.

Listing 3-24. Printing Differencing Removed Forecasted Time Series

```
predictions_ARIMA_log = pd.Series(data_log.ix[0], index=data_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_
cumsum,fill_value=0)
predictions_ARIMA_log.head()
```

Output

```
2016-01-04    7.607213
2016-01-05    7.607614
2016-01-06    7.607850
2016-01-07    7.609631
2016-01-08    7.613261
dtype: float64
```

The next step was to remove log transformation from the forecasted time series. Recalling his learnings from his Calculus 1 course in college he knew that the exponential acts as the inverse of log base 10. In Listing 3-25 he planned to do that, and rate the forecasted series by the aid of evaluation metrics.

Listing 3-25. Evaluating the Forecasted Series vs. the Original One

```
def mean_forecast_err(y, yhat):
    return y.sub(yhat).mean()

def mean_absolute_err(y, yhat):
    return np.mean((np.abs(y.sub(yhat).mean()) / yhat)) # or
    percent error = * 100

def rmse(y, yhat):
    return np.sqrt(sum((yhat-y)**2)/len(y))

predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(data_train['Close_Adj'])
plt.plot(predictions_ARIMA)

plt.title('RMSE: %.4f | MFE: %.4f | MAE: %.4f'% (
        rmse(data_train['Close_Adj'], predictions_ARIMA), mean_forecast_
            err(data_train['Close_Adj'], predictions_ARIMA), mean_absolute_
            err(data_train['Close_Adj'], predictions_ARIMA)))
```

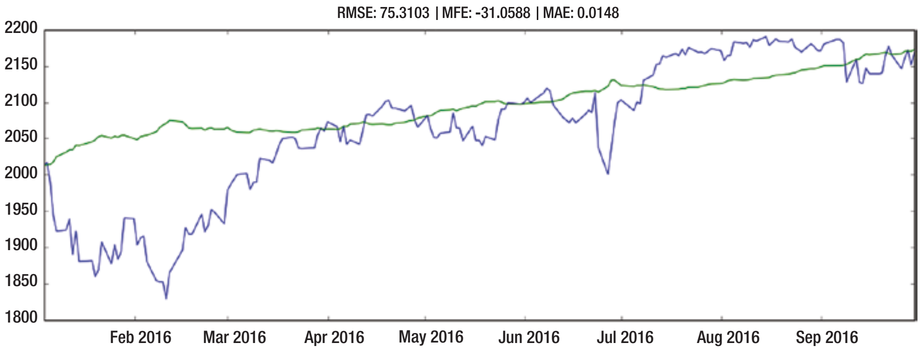


Figure 3-21. *Plotting forecasted series vs. the original time series*

In Listing 3-24, David at first initialized a separate method for each of the evaluation metrics. He then removed the log from the forecasted series. In the end he validated the forecasted series by comparing it to the real one, both statistically and graphically. The test statistics of RMSE, MFE, and MAE revealed that this model had successfully forecasted correctly, which was good news for David. Forecasted plot is represented by the green line, with the original time series represented by a blue one. David was surprised to see in Figure 3-21 that the forecasted time series resembled that of a linear regression line.

Hence David decided to apply regression to the near to linear forecasted series seen in Figure 3-21. He planned to train the regression line from the linear forecasted series, predict for data from October to December 2016, and then cross-validate it from the real values for that time period. He did so by writing the code in Listing 3-26.

Listing 3-26. Predicting and Evaluating Future Time Series Using a Linear Regression Model

```
regr = LinearRegression()

x_train = [[x] for x in range(len(data_train))]
y_train = [[y] for y in list(data_train['Close_Adj'])]

x_test = [[z] for z in range(x+1, x + 1+len(data_test))]

regr.fit(x_train, y_train)

y_pred = regr.predict(x_test)

explained_variance_score(data_test['Close_Adj'], y_pred)

plt.scatter(range(len(data_test)), data_test['Close_Adj'],  color='black')
plt.plot(range(len(data_test)), y_pred, color='blue',
             linewidth=3)
plt.ylabel('Concrete strength')
```

Output

```
0.63472586609437109
```

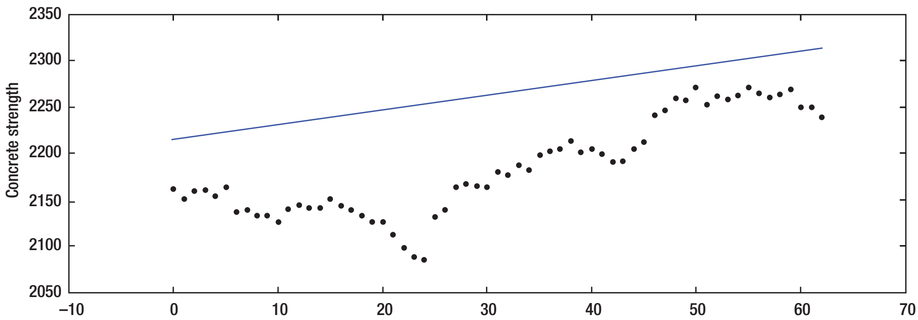


Figure 3-22. *Plotting forecasted series along with the predicted regression line for Oct-Dec 2016*

David was happy to see the regression line in Figure 3-22 fairly capture the trend within the dataset. He performed model training on the forecasted time series and evaluated it against the original time series of Yahoo stock. He also made sure to calculate

the explained variance score as well to see how much variance of the time series is correctly captured by the best fit line. The value came out to be 0.63, which indicated that 63% of the variance within the time series was captured by the linear regression line.

Given the results he had achieved within a short time span, David was optimistic that he could achieve more if he had more days to spare. However, he wants you to come on board and help him experiment with different time series forecasting strategies. His rationale is that he further wants to improve the accuracy of the forecasted series, as it is the input being fed into the regression line. Hence, he has left the following exercises for you to solve.

EXERCISES

1. Repeat the above steps by applying decomposition instead of differencing. Did it improve the forecast accuracy?
2. Repeat the exercise by changing the parameter values of p, d, and q for the ARMA, and ARIMA models. Then compute the test statistics to see if a given combination of these parameters yields a better forecast accuracy.

Applications of Time Series Analysis

Sales Forecasting

Corporate organizations are concerned about the expected sales for the next period. Sales forecasts help them make rational decisions to achieve the targets at hand. Sales usually have a strong element of seasonality in them. Consider, for example, that we are forecasting the sales in dollar amount of groceries on a daily basis. We will see a seasonality that repeats monthly with the hike in sales coming in the first week of every month. Moreover, the change in habits of people (i.e., spending and consumption) can instill a positive and negative trend in the series.

Weather Forecasting

This is the de facto application for time series analysis. Weather forecasting comes along with a lot of elements other than just the temperature. Forecasts are done for amount of rainfall, humidity, and wind speed as well.

Unemployment Estimates

Time series analysis is also used to estimate unemployed people in our economy. Unemployment is strongly affected by change in foreign direct investment, the security situation of a country, change in rate of skilled labor, and so on. These estimates are highly important as they affect the government's expenditures for social and relief programs. Moreover, they help the government estimate the amount of change in direct and indirect tax as consumption levels increase due to the creation of employment.

Disease Outbreak

Nongovernmental organizations (NGOs) and social organizations are interested in forecasting the outbreak of a disease across the globe. This is important for them so that they can launch awareness programs and prevention strategies in regions highly susceptible to a particular disease or epidemic.

Stock Market Prediction

Stocks fluctuate a lot on a short-term basis during trading hours, which makes it difficult for investors to forecast the point in time when they can benefit the most. However, the problem becomes more critical when investors invest in a portfolio of stocks and are worried about the return they should expect from it in the future or adjust a stock in order to receive returns in their favor. This is where time series analysis comes in handy.

David didn't forget to summarize the methodology incorporated to reach the end result. He recalled having first defined the problem statement (i.e., predict Yahoo stock with a minimal residual). At first he had to understand what a time series is and then he validated the test of stationary against the time series object. Transformation and differencing were done to make the series object stationary. Then he applied the time series model and chose the one with the lowest RSS. This forecasted time series was then scaled back and used as an input to the linear regression line. The regression line enabled the extrapolation of future values and cross-validation was done to determine the variance explained by the best fit line.

David now planned to incorporate this model to predict Janice's expected returns and to avoid a major setback due to prediction inaccuracies.

CHAPTER 4

Clustering

The goal of this chapter is to solve real-world problems with the aid of supervised and unsupervised learning algorithms. First we will start off with the concept of clustering, determine how to organize the data, decide upon the number of components, and then end up seeing if the cluster outputs make any intuitive sense.

Note This book incorporates **Python 2.7.11** as the de facto standard for coding examples. Moreover, you are required to install it for the *Exercises*.

In this chapter we will be using the **Accepted Papers dataset** for clustering coding examples. This data dump can be downloaded from

https://archive.ics.uci.edu/ml/datasets/AAAI+2014+Accepted+Papers

Case Study: Determination of Short Tail Keywords for Marketing

Ross, the marketing director of an artificial intelligence (AI) conference, had to report to the board with recommendations for short tail keywords for the marketing of the 2015 conference. He also had to come up with a visualization of research papers submitted so far, grouped by their paper type.

The conference is run by a nonprofit scientific society that ensures advancements in the domains of artificial intelligence. It does so by enhancing public understanding of the domain, as well as by providing researchers with a platform for them to present their findings in yearly conferences.

In order to come up with a solution for the problem at hand, Ross started off by going through the web site. However, the magnitude of content from the research papers was too much for him to make an analysis.

D. Haroon, *Python Machine Learning Case Studies*, DOI 10.1007/978-1-4842-2823-4_4

He thus knocked on Ted's door to find out if any data was available to start off the analysis. The meeting with Ted, who was the director of the data warehousing department, gave Ross a glimpse of hope. As Ted recalled:

> *Ross came up to me with this request, but the problem was that we were going through a major overhaul of integrating SAP in/out processes. But fortunately we had some data dumps available in storage. I asked Ross for the conference year he wanted the data for, and boom we had the data dump for that year.*

Ross loaded the data dump in Microsoft Excel and started looking through the features. He thought that one strategy could be to group the papers by keywords or groups. He could then run separate ads on social media and use SEO (search engine optimization) to target the respective keywords within them. However, there were too many distinct values within keywords and group features, and thus they would result in many clusters of SEO keywords. His strategy was to come up with a maximum of ten different SEO keyword groups and let them run for several days to reap benefits. He had heard something about clustering, where you cluster the data into a fixed number of clusters, and so he decided that clustering would be his starting point.

Now that he had a path to follow, he had some questions in mind. How many clusters would be optimal from the data provided? Given that the data is small, can any machine learning algorithm be a best fit to it? How should he visually present the data and, ultimately, the findings?

Ross believed that understanding the features of data provided by Ted would prove beneficial later on down the path. He felt that understanding them would help him decide upon the best approach for solving the problem at hand. Hence he compiled the data dictionary in Table 4-1.

Table 4-1. *Data Dictionary for the Accepted Papers Dataset*

Feature name	Description
Title	Title of the paper
Authors	Author(s) of the paper
Groups	Author-selected, high-level keyword(s)
Keywords	Author-generated keywords
Topics	Author-selected, low-level keywords
Abstracts	Paper abstracts

While looking at Table 4-1, Ross deduced that all of the features were in a string format. Moreover, he noticed that paper manuscripts are not part of the dataset and abstracts were provided instead.

Ross had recently completed the Python specialization offered by the University of Michigan on Coursera. Hence he was confident of his Python skills to nail the job. Before starting with the analysis, he decided to load all the relevant packages into the memory as shown in Listing 4-1.

Listing 4-1. Importing Packages Required for This Chapter

```
%matplotlib inline

import operator
import itertools
import numpy as np
import pandas as pd
from ggplot import *
import seaborn as sns
import matplotlib as mpl
from sklearn import mixture
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
from sklearn.decomposition import PCA
from wordcloud import WordCloud, STOPWORDS
from scipy.spatial.distance import cdist, pdist
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.metrics import euclidean_distances, silhouette_score

rcParams['figure.figsize'] = 15, 5
```

Though Ross knew the features and their descriptions, he was curious to know the contents within.

Features' Exploration

Before moving forward, it was time for him to lay out his strategy. After putting much thought into it, he finally came up with a plan that could potentially work. The plan was to initially break the research papers into segments so that he could focus on each one of them by means of the targeted short tail and long tail keywords. He then looked at the features in Table 4-1 to filter out the ones that resonated to his approach. He believed that the Authors field was irrelevant and that the Abstract field was too detailed in nature. Hence he narrowed his search to just four features (Title, Groups, Keywords, and Topics). He filtered the features by writing down the code snippet in Listing 4-2.

Listing 4-2. Reading the Data in the Memory, and Subsetting Features

```
data_train = pd.read_csv('examples/[UCI] AAAI-14 Accepted Papers - Papers.csv')
data_train = data_train[['title', 'groups', 'keywords', 'topics']]
```

Ross was now interested to know how big the dataset was (i.e., how many research papers were available). Moreover, he was interested in taking a sneak peek of the dataset. He wrote the script in Listing 4-3 for that purpose.

Listing 4-3. Printing the Size of the Dataset and Printing the First Few Rows of the Dataset

```
print len(data_train)
data_train.head()
```

Output

```
398
```

Table 4-2. *Print of Observations of the Dataset*

	title	groups	keywords	topics
0	Kernelized Bayesian Transfer Learning	Novel Machine Learning Algorithms (NMLA)	cross-domain learning\ndomain adaptation\nkern...	APP: Biomedical / Bioinformatics\nNMLA: Bayesi...
1	"Source Free"Transfer Learning for Text Class...	AI and the Web (AIW)\nNovel Machine Learning A...	Transfer Learning\nAuxiliary Data Retrieval\nT...	AIW: Knowledge acquisition from the web\nAIW: ...
2	A Generalization of probabilistic Serial to Ra...	Game Theory and Economic Paradigms (GTEM)	social choice theory\nvoting\nfair division\ns...	GTEP: Game Theory\nGTEP: Social Choice / Voting
3	Lifetime Lexical Variation in Social Media	NLP and Text Mining (NLPTM)	Generative model\nSocial Networks\nAge Prediction	AIW: Web personalization and user modeling\nNL...
4	Hybrid Singular Value Thresholding for Tensor...	Knowledge Representation and Reasoning (KRR) \n...	tensor completion\nlow-rank recovery\nhybrid s...	KRR: Knowledge Representation (General/Other) \...

From the output of Listing 4-3, Ross noticed that the dataset has data on approximately 400 research papers. Moreover, he found validation for his deduction that all of the features within the dataset are in a string data format. He also noticed that a given research paper can fall into more than one group and keyword.

Ross was on his way home, on the subway, waiting for the next train to arrive, when he ran into his college mate Matt, with whom he had partnered for community work. After college Matt had made a career in the domain of data analytics. Ross thus brought up the problem he was trying to solve. After listening diligently Matt advised Ross to decide whether he was going to use supervised learning or unsupervised learning before moving onto model selection. It was a brief conversation as Matt had to go, and it left Ross with many questions about this concept. It was not until the next morning that Ross returned to the office and did research on supervised and unsupervised learning methods.

Supervised vs. Unsupervised Learning

Machine learning algorithms are usually divided into supervised and unsupervised learning.

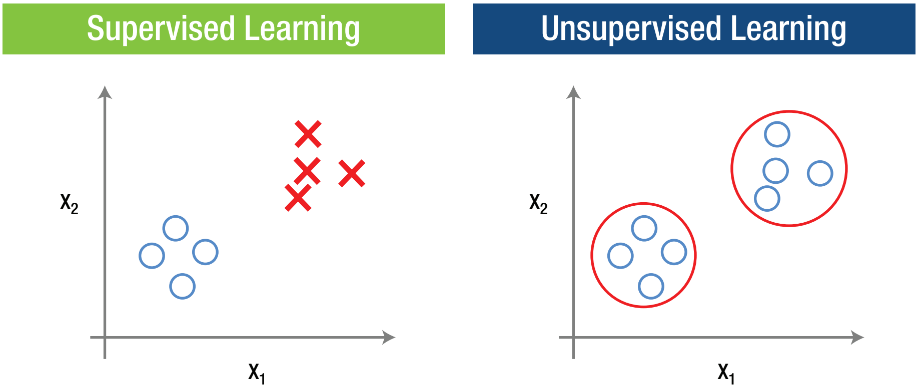


Figure 4-1. *Supervised vs. unsupervised learning*

Ross came up with an explanation for both these methods.

Supervised Learning

As the name suggests, supervised learning algorithms require supervision for them to train the model. This supervision is usually necessary in the case of classification where we have labeled data on which we train the model for it to predict the labels of the unseen data. Here, the supervision is by means of the label provided with each and every observation (i.e., supervising the learning process). Examples include classification for discrete predictors and regression for the continuous ones.

Unsupervised Learning

Unsupervised learning algorithms require no supervision from the data while training the model. A prime example of this is clustering, which discovers the labels without any supervision needed. These discovered labels then become the basis for classifying any new unseen data. Another example of unsupervised learning is the rule of association, which entails the concepts of complement and substitute. Complement refers to a phenomenon whereby if a shopper buys X then, with a high degree of certainty, he will buy Y as well. Substitute refers to a behavior whereby a shopper will buy either X or Y.

Other examples include anomaly detection, method of moments (e.g., mean and covariance), independent component analysis, and principle component analysis (PCA).

Ross had to decide on which of the two is suitable for his problem at hand and the available data. From the data in Table 4-2, he noticed that the dataset provided no labeled data. This meant that he had to predict labels from scratch using unsupervised learning.

Ross was assuming that clustering, being an unsupervised learning method, could help him find the correct segments to meet the marketing goals. However, he wasn't too sure if clustering was the right approach to pursue. Hence he came up with the following material on the topic.

Clustering

Cluster analysis refers to the grouping of observations so that the objects within each cluster share similar properties, and properties of all clusters are independent of each other. Cluster algorithms usually optimize by maximizing the distance among clusters and minimizing the distance between objects in a cluster. Cluster analysis does not complete in a single iteration but goes through several iterations until the model converges. Model convergence means that the cluster memberships of all objects converge and don't change with every new iteration. Some clustering algorithms don't ask for the number of clusters/components, and come up with the number of clusters that statistically make more sense. However, a huge chunk of clustering algorithms prompt the user upfront for the number of clusters/components he desires in the output. It is important to understand that clustering, just like classification, is used to segment the data; however, these groups are not previously defined in the training dataset (i.e., they are unlabeled data).

Different algorithms deploy different techniques for the computation of clusters. Some of those techniques are as follows:

- Partitioning: Groups data into given number of clusters while optimizing for the objective (e.g., distance).
- Hierarchical: Groups data into a hierarchy of clusters. These hierarchies are formed top-down or bottom-up.
- Grid-based: Divides the data into hyper-rectangle cells, discards low-density cells, and combines high-density cells to form clusters.

A good clustering algorithm satisfies the following requirements:

- Within-cluster similarity and between-cluster dissimilarity
- Can deal with training dataset which is:
    - Of a high dimension
    - Affected by noise and outliers

Ross now had some knowledge of supervised and unsupervised models; however, he was not sure if clustering could be applied to data that is textual in nature. Moreover, he knew that the greater the number of features fed in, the higher the chances to discover features that influence during the cluster discovery process. Hence he had to figure out a way to convert textual data into numeric form and use the existing four features to generate more features.

Data Transformation for Modeling

Ross spent hours searching for a solution to this mystery. He felt disappointed as he had expected this approach to work and yield results. To make it easy, he narrowed down his search to one feature (i.e. groups) and started pondering on how he could recode it into a numeric form. It was late at night, but Ross couldn't sleep, and then an idea struck him. Why not convert the "groups" feature into Boolean, such that a research paper that falls within a group will have 1 marked next to it, and 0 otherwise. The data representation he could think of would be a matrix where columns will represent the different groups and rows will represent the research papers.

He knew that a research paper could fall into one group. Groups for a given research paper were separated by a delimiter (i.e., \n) in the dataset. Hence he first decided to separate groups by dividing the delimiter in between so that if a research paper falls within three groups it will be represented by three different observations (i.e., rows) in the dataset.

Listing 4-4. Stretching the Data Frame Row-wise as a Function of Groups

```
s = data_train['groups'].str.split('\n').apply(pd.Series, 1).stack()
s.index = s.index.droplevel(-1)
s.name = 'groups'
del data_train['groups']
data_train = data_train.join(s).reset_index()
```

In addition to separating groups, Ross also decided to add a new feature "flags," and assign a value of 1 to all the rows. As per his notion, a value of 1 will denote that the research paper in that row belongs to the group in the "groups" column.

Listing 4-5. Adding New Variable for Group Membership

```
data_train['flags'] = pd.Series(np.ones(len(data_train)),
index=data_train.index)
data_train.head()
```

Table 4-3. *Print of Observations of the Dataset After Data Frame Transformation*

title	keywords	topics	groups	flags
Kernelized Bayesian Transfer Learning	cross-domain learning\ndomain adaptation\nkern...	APP: Biomedical / Bioinformatics\nNMLA: Bayesi...	Novel Machine Learning Algorithms (NMLA)	1.0
"Source Free"Transfer Learning for Text Class...	Transfer Learning\nAuxiliary Data Retrieval\nT...	AIW: Knowledge acquisition from the web\nAIW: ...	AI and the Web (AIW)	1.0
"Source Free"Transfer Learning for Text Class...	Transfer Learning\nAuxiliary Data Retrieval\nT...	AIW: Knowledge acquisition from the web\nAIW: ...	Novel Machine Learning Algorithms (NMLA)	1.0
A Generalization of probabilistic Serial to Ra...	social choice theory\nvoting\nfair division\ns...	GTEP: Game Theory\nGTEP: Social Choice / Voting	Game Theory and Economic Paradigms (GTEM)	1.0
Lifetime Lexical Variation in Social Media	Generative model\nSocial Networks\nAge Prediction	AIW: Web personalization and user modeling\nNL...	NLP and Text Mining (NLPTM)	1.0

Having expanded a research paper as a function of the group it belongs in, and having associated a member flag to it, Ross knew that the time had come to convert this data structure into a matrix. For that he defined the function in Listing 4-6.

Listing 4-6. Adding a Function for Matrix Creation

```
def matrix_from_df(data_train):

    matrix = data_train.pivot_table(index = ['title'], columns=['groups'],
    values='flags')
    matrix = matrix.fillna(0).reset_index()
    x_cols = matrix.columns[1:]
    return matrix, x_cols
```

Over the years, Ross had become proficient in pivot tables within Microsoft Excel. He thus decided to use the same methodology in Listing 4-6 for the data frame to matrix transformation. Hence he initialized a method in Listing 4-6 whereby he converted the training data into a pivot with title as an index, each group represented by a separate column, having values from the “flags” feature. Then he placed 0 where the research paper in the index didn't fall within the realm of a given group. He decided to bring the matrix live by using the method defined in Listing 4-6.

Listing 4-7. Retrieve Matrix and x cols from the matric_from_df Method

```
matrix, x_cols = matrix_from_df(data_train)
matrix.head()
```

Table 4-4. *Initial Observations of the Matrix*

groups	title	AI and the Web (AIW)	Applications (APP)	Cognitive Modeling (CM)	Cognitive Systems (CS)	Computational Sustainability and AI (CSAI)	Game Playing and Interactive Entertainment (GPIE)	Game Theory and Economic Paradigms (GTEP)	Heuristic Search and Optimization (HSO)	Human-Computation and Crowd Sourcing (HCC)	...
0	“Source Free” Transfer Learning for Text Class...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
1	A Characterization of the Single-Peaked Single...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...
2	A Computational Method for (MSS, CoMSS) Partiti...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...

Ross was thrilled to see the output come out as expected. To test his understanding, he picked a research paper from Table 4-4 at random. He chose "A Computational Method for (MSS.CoMSS)." By looking at the matrix in Table 4-4, Ross inferred that the paper fell in group HSO and didn't have membership in any of the other groups.

Now that he knew what clustering was, and having transformed the data into a form friendly for applying it, it was time for him to apply clustering modeling. However, first he had to determine the evaluation metrics by which he could evaluate the goodness of a clustering model. Contrary to the techniques available for evaluating regression and classification models, not many techniques are available to statistically evaluate the goodness of a clustering model. The reason is that in regression and classification, labeled data is already available on which the model is trained. However, this is not the case with clustering.

Metrics of Evaluating Clustering Models

Having done some research, Ross learned that there is no one way of to evaluate the output of a clustering model. He came across two approaches for making it work. The first was more of a technical nature. It deems a clustering model good enough if maximum variance is explained within each of the clusters, objects within the clusters share similar properties, and clusters are further away from each other. The second approach indicated the model to be strong if the cluster definitions make intuitive sense.

Ross now had a clear understanding of clustering and methodologies to evaluate its goodness. Hence he started to look for clustering models that work best in related applications. After thorough research, he came up with several clustering models.

Clustering Models

Ross decided to start with the de facto clustering algorithm (i.e., k-means clustering). To Ross, understanding the methodology by which clusters converge was pivotal before applying it to the dataset in hand. Hence he compiled an explanation of the k-means clustering algorithm.

k-Means Clustering

k-means clustering partitions the data space into Voronoi cells representation. This transformation divides the data observations into k-clusters in which each of the observations belongs to the cluster with the nearest mean. k-means clustering is done as follows:

1. k-centroids are chosen randomly.
2. Each of the observations is binded with the closest centroid.
3. New centroids for each of the clusters are recomputed by taking the mean of the observations lying within each cluster.
4. Step 2 is repeated again.

You might note that steps 1 and 3 resemble each other, the difference being that in step 1, centroids are chosen at random, whereas in step 3 they are calculated by taking the mean of observations within each of those clusters.

Then steps 2 and 4 are a repetition of the same step. Steps 3 and 4 are iterated until the clusters converge—that is, cluster memberships remain constant for all observations in the dataset. This maximizes the distance between clusters and minimizes the distance among the observations within each cluster. k-means clustering does not always find the most optimal configuration while minimizing the global objective function. The clustering algorithm is highly sensitive to how cluster centers are initially selected.

Ross now fully understood how convergence is performed in a k-means clustering algorithm. However, he was clueless on as to what value of k (i.e., number of clusters he should specify upfront) would be optimal for the data at hand. He recalled having seen an article on k-means clustering on Linkedin. Hence he searched for the person who had posted that article and then messaged him asking for his assistance in this regard. His connection, Pollack, suggested that Ross use the following methods:

- Elbow method
- Variance explained
- BIC score

He started off by looking at how the Elbow method works and then applied it to find the number of clusters optimal to the model.

Elbow Method

The Elbow method in Ross's terms is the percentage of variance explained as a function of the number of clusters. It determines how much marginal variance is contributed by a newly added cluster. A point of interest is that the first cluster will explain maximum variance with marginal gains dropping on the addition of every new cluster. Elbow will be the point where a new cluster will result in a considerable marginal drop, that being the optimal number of clusters. Ross decided to apply the Elbow method to the matrix transformed dataset.

Listing 4-8. Applying Elbow Method and Variance Explained on Data Matrix

```
matrix, x_cols = matrix_from_df(data_train)
X = matrix[x_cols]

K = range(1,50)
KM = [KMeans(n_clusters=k).fit(X) for k in K]
centroids = [k.cluster_centers_ for k in KM]

D_k = [cdist(X, cent, 'euclidean') for cent in centroids]
dist = [np.min(D,axis=1) for D in D_k]
avgWithinSS = [sum(d)/X.shape[0] for d in dist]
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(K, avgWithinSS, 'b*-')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('Average within-cluster sum of squares')
plt.title('Elbow for KMeans clustering')
```

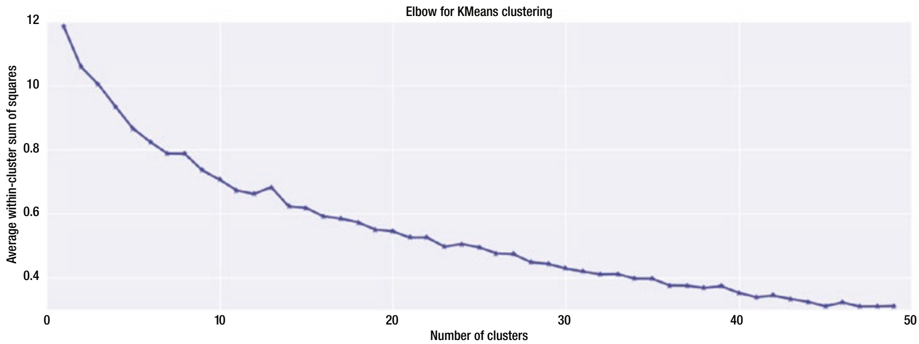


Figure 4-2. *Elbow method curve as a function of number of clusters*

In Listing 4-8, Ross recalled training 49 models of k-means clustering, each having a different k-value ranging from 1 to 50. He then fetched the cluster centers for those clustering models and determined the cluster memberships for each research paper model by model. Finally, for each model, he computed the sum of all research papers and divided it by the total number of research papers to get the average within cluster sum of squares. He then plotted the cluster sum of squares for all of the 49 cluster models in Figure 4-2 to formulate the Elbow method.

While looking at Figure 4-2, Ross deduced the elbow to occur at k = 9. However, he wasn't sold until and unless more than one method doesn't converge on the same number of clusters. Thus he moved toward understanding variance explained.

Variance Explained

The percentage of variance explained, or in other words F-test, is the ratio of group variance to the total variance. Here again the elbow will determine the optimal number of clusters for the k-means clustering model. Ross put variance explained into practice by writing the code snippet in Listing 4-9.

Listing 4-9. Applying Elbow Method and Variance Explained to Data Matrix

```
matrix, x_cols = matrix_from_df(data_train)
X = matrix[x_cols]

K = range(1,50)
KM = [KMeans(n_clusters=k).fit(X) for k in K]
centroids = [k.cluster_centers_ for k in KM]

D_k = [cdist(X, cent, 'euclidean') for cent in centroids]
dist = [np.min(D,axis=1) for D in D_k]

wcss = [sum(d**2) for d in dist]
tss = sum(pdist(X)**2)/X.shape[0]
bss = tss-wcss

kIdx = 10-1

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(K, bss/tss*100, 'b*-')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('Percentage of variance explained')
plt.title('Elbow for KMeans clustering')
```

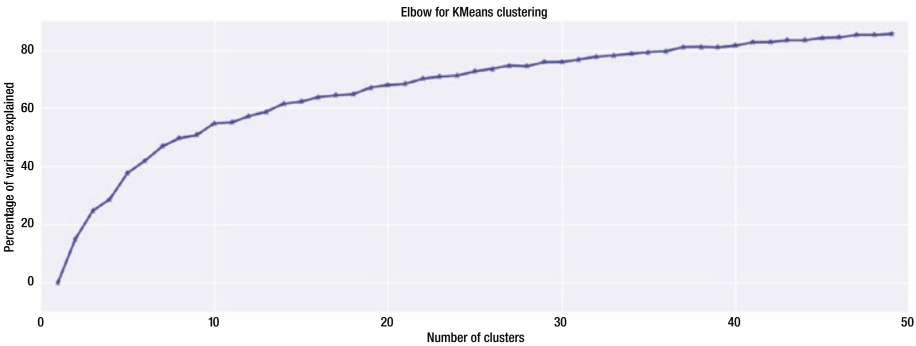


Figure 4-3. *Explained variance curve as a function of number of clusters*

Ross recalled having explained Listing 4-9 in the following words:

> *After getting the cluster memberships of each research paper cluster by cluster, the distances were then used to compute the within cluster sum of square. Next, distances were computed among the columns within each research paper to compute the total sum of squares. Finally, within the cluster sum of square was subtracted from total sum of squares to get the between sum of squares. Ratio of between sum of squares and total sum of squares was taken to plot in Figure 4-3.*

To Ross Figure 4-3 looked like an exponential cumulative distribution function. The variance explained was seemingly smooth and kept increasing with the number of clusters. This made it difficult for Ross to figure out the elbow. After due consideration, Ross noticed the gradient to start smoothing from k = 9, similar to what he had observed with the Elbow method.

However, he didn't stop there and continued looking for methods to help determine the optimal k, and within a short time he was exposed to the Bayesian Information Criterion (BIC) score.

Bayesian Information Criterion Score

BIC is another technique to choose the best model out of a finite set of models. The model with the lowest BIC score is deemed to be the winner. When training the model a high probability of increasing the likelihood of getting accurate clusters exists by adding extra parameters; however, he knew that doing so would overfit the model. BIC overcomes this problem by adding a penalty term on number of parameters in the model. BIC works on the assumption that the sample size should be much larger than the number of parameters in the model. Hence BIC won't be suitable for complex models working on high-dimensional data.

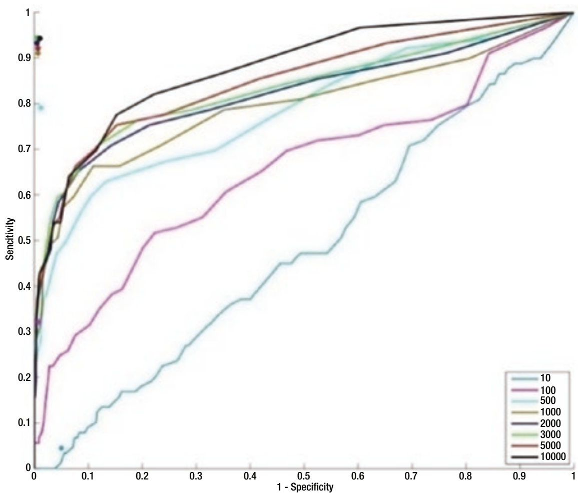


Figure 4-4. *Sample BIC score plot*

The knowledge base on BIC score made Ross drop it as an option, because that BIC score is suitable for low-dimensional data, and therefore the matrix transformation meant that now the data had (number of groups + 1) dimensions, making it high-dimensional data.

He was curious to know if a method existed to determine the optimal number of clusters, a method which resonates to how a k-means cluster works. In other words, Ross wanted to find a method that determines the optimal number of clusters on which intercluster distances are the maximum and intracluster distances are at a minimum. His search made him look into research papers, where he found the Silhouette method.

Silhouette Score

Silhouette score is used to measure how close observations within a cluster are (i.e., intra-cluster distance (a)), and how distinct the clusters are from each other (i.e., mean nearest cluster distance (b)). The Silhouette coefficient is calculated as follows:

$$(b-a) / \max(a, b)$$

The best value is 1 and the worst is -1. A coefficient value of 0 indicates overlapping clusters. The catch for Ross was now to select the number of clusters that yield the maximum Silhouette score. Without wasting any time he applied the Silhouette method on k-means models, within the cluster size range of 2 to 30.

Listing 4-10. Plotting Silhouette Score Plot from the Data Matrix

```
s = []

for n_clusters in range(2,30):
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X)

    labels = kmeans.labels_
    centroids = kmeans.cluster_centers_

    s.append(silhouette_score(X, labels, metric='euclidean'))

plt.plot(s)
plt.ylabel("Silouette")
plt.xlabel("k")
plt.title("Silouette for K-means cell's behaviour")
sns.despine()
```

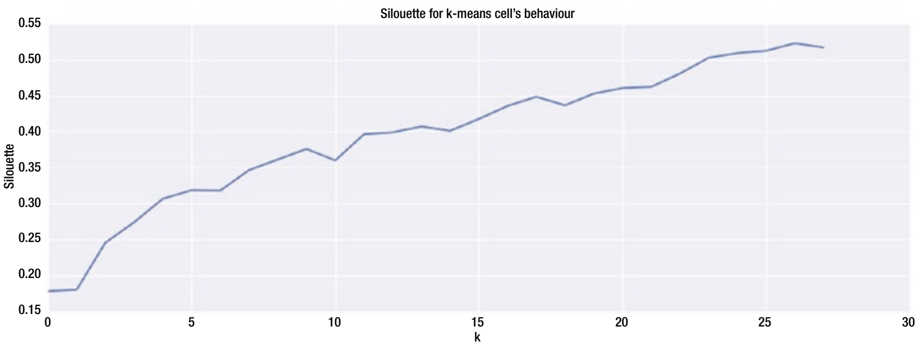


Figure 4-5. *Silhouette coefficients plot as a function of number of clusters*

Ross was puzzled to see the plot in Figure 4-5 as the Silhouette coefficient seemed to increase continuously with the increasing number of clusters. As the theory suggested the optimal number of clusters to be the point where the Silhouette coefficient is the highest, he couldn't realistically choose 27 to be the right number of clusters for the following reasons:

- 27 clusters will be too many for a data matrix having 398 observations in total.
- He realistically couldn't market 27 different segments within the limited amount of time he had.

Hence he decided to proceed with nine clusters, which he determined from the Elbow method (i.e., Figure 4-3) and variance explained (i.e., Figure 4-4).

Applying k-Means Clustering for Optimal Number of Clusters

Ross applied k-means clustering with a cluster size of 9 in Listing 4-11.

Listing 4-11. Training k-means Model for Cluster Size of 9

```
matrix, x_cols = matrix_from_df(data_train)
X = matrix[x_cols]

cluster = KMeans(n_clusters = 9, random_state = 2)
matrix['cluster'] = cluster.fit_predict(X)
matrix.cluster.value_counts()
```

Output

```
0    81
1    78
2    32
3    38
4    27
5    44
6    36
7    30
8    30

Name: cluster, dtype: int64
```

In Listing 4-11, Ross decided to print the number of research papers that fall within each of the nine clusters. He noticed that except for two clusters, which had 80 observations each, the remaining seven clusters had 30 observations each on average.

Ross was interested to see what these clusters would look like visually. He was aware that dimensions within the dataset are a function of the number of distinct groups. Ross knew that no package within Python would be sufficient to plot such a high-dimensional figure. Even if he were successful in plotting it, he was aware that the figure would be too complex to extract any intuitive sense. Hence he started looking for a method by which he could somehow reduce the number of dimensions. He didn't have to wait for long before he was exposed to principle component analysis (PCA).

Principle Component Analysis

PCA converts a set of observations that are highly correlated into a set of linearly uncorrelated variables called principle components by an orthogonal transformation. This transformation is done such that the first component has the highest variance.

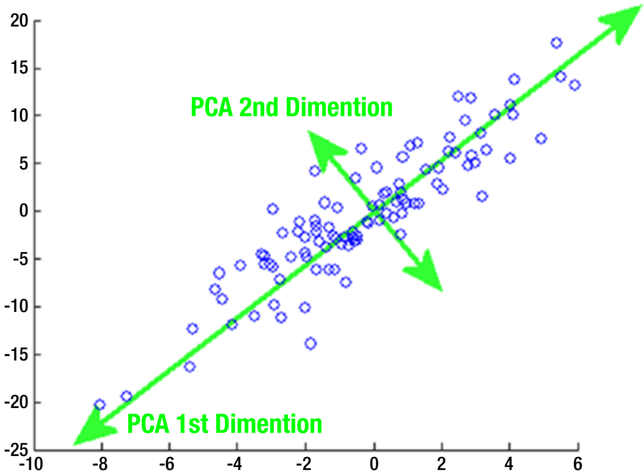


Figure 4-6. *Illustration of PCA*

Subsequent components have lower variances, keeping the constraint under consideration that it is orthogonal to the preceding components as depicted in Figure 4-6. PCA is highly sensitive to relative scaling of the data. PCA can be thought of as a dimension reduction technique which reduces high-dimensional data to a fixed number of components.

Having read the description, Ross was sure that with dimension reduction he was doing the right thing. Hence without wasting any further time he wrote the code in Listing 4-11 for reducing the group dimensions to two dimensions (i.e., x, and y). He decided to convert it into two dimensions to make it easy for him to plot the clusters on a two-dimensional axis.

Listing 4-12. Using PCA to Transform Group-Related Features into Two Components

```
pca = PCA(n_components=2)
matrix['x'] = pca.fit_transform(matrix[x_cols])[:,0]
matrix['y'] = pca.fit_transform(matrix[x_cols])[:,1]
matrix = matrix.reset_index()

customer_clusters = matrix[['title', 'cluster', 'x', 'y']]
customer_clusters.head()
```

Table 4-5. *Print of Clusters Along with Two Newly Created Components Using PCA*

groups	title	cluster	x	y
0	"Source Free" Transfer Learning for Text Class...	1	0.615810	-0.060295
1	A Characterization of the Single-Peaked Single...	8	-0.756838	0.971322
2	A Comutational Method for (MSS, CoMSS) Partiti...	4	-0.287956	-0.216148
3	A Control Dichotomy for Pure Scoring Rules	2	-0.521295	0.570206
4	A Convex Formulation for Semi-supervised Multi...	3	0.198578	-0.129668

Having done the transformation in Table 4-5, Ross now had to plot that on a scatter plot for which he wrote the code in Listing 4-13.

Listing 4-13. Plotting Clusters in a Two-Dimensional Space

```
cluster_centers = pca.transform(cluster.cluster_centers_)
cluster_centers = pd.DataFrame(cluster_centers, columns=['x', 'y'])
cluster_centers['cluster'] = range(0, len(cluster_centers))

plt.scatter(customer_clusters['x'], customer_clusters['y'], s = 20,
c=customer_clusters['cluster'])
plt.scatter(cluster_centers['x'], cluster_centers['y'], s = 150, c=cluster_
centers['cluster'])
```

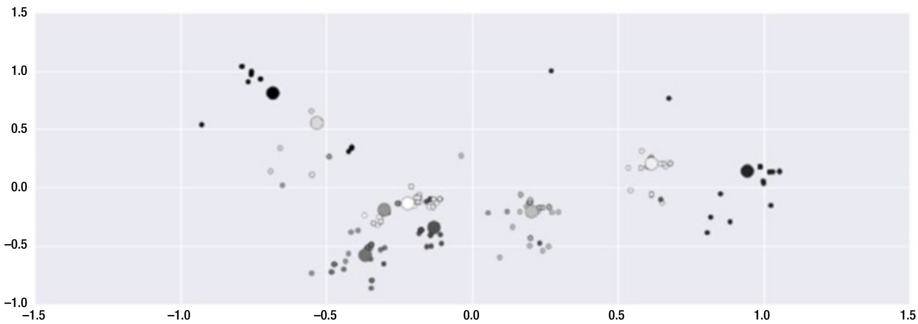


Figure 4-7. *Clusters in two-dimensional space*

While writing down the code in Listing 4-13 Ross made sure to assign a different color label to each cluster observation so that he could visually see their cluster memberships in Figure 4-7. Moreover, he made sure to represent the cluster centers as well. Cluster memberships in Figure 4-7 seemed to him to be distinct and non-overlapping. However, recalling his initially laid objective of finding the keywords associated with each segment, he had to merge the dimension-reduced data into the original data (i.e., the one before matrix transformation). For that purpose he wrote down the script in Listing 4-14.

Listing 4-14. Merging Matrix into the Original Data Frame

```
customer_clusters.columns.name = None
df = data_train.merge(customer_clusters, on='title')
df.head()
```

Table 4-6. *Print of Observations of the Merged Data Frame*

title	keywords	topics	groups	flags	cluster	x	y
Kernelized Bayesian Transfer Learning	cross-domain learning\ndomain adaptation\nkern...	APP: Biomedical / Bioinformatics\nNMLA: Bayesi...	Novel Machine Learning Algorithms (NMLA)	1.0	1	0.613870	0.245408
"Source Free"Transfer Learning for Text Class...	Transfer Learning\nAuxiliary Data Retrieval\nT...	AIW: Knowledge acquisition from the web\nAIW:...	AI and the Web (AIW)	1.0	1	0.615810	-0.060295
"Source Free"Transfer Learning for Text Class...	Transfer Learning\nAuxiliary Data Retrieval\nT...	AIW: Knowledge acquisition from the web\nAIW:...	Novel Machine Learning Algorithms (NMLA)	1.0	1	0.615810	-0.060295
A Generalization of Probailistic Serial to Ra...	social choice theory\nvoting\nfair division\ns...	GTEP: Game Theory\nGTEP: Social Choice / Voting	Game Theory and Economic Paradigms (GTEP)	1.0	2	-0.521295	0.570206
Lifetime Lexical Variation in Social Media	Generative mode\nSocial Networks\nAge Prediction	AIW: Web personalization and user modeling\nNL...	NLP and Text Mining (NLPTM)	1.0	0	-0.183192	-0.090091

Ross made sure to print the first few observations of this merged data object. He was successful in merging the two data representations as now the data had title, keywords, topics, and gender features from the original dataset and features of flags, cluster, x, and y from the dimension-reduced dataset. He now had to use this merged dataset to generate keywords for each of the segments. He decided to do so by using a wordcloud. A wordcloud will show all words associated with a given cluster, with the word font sizes representing their frequency of occurrence within the pivotal feature. He planned to keep the pivotal feature arbitrary, and this pivotal feature would be used to supply words to the wordcloud. He started off by writing a method in Listing 4-15 to generate wordclouds.

Listing 4-15. Creating Function to Generate Wordcloud

```
def wordcloud_object(word_string):

    FONT_ROOT = './fonts/'
    wordcloud = WordCloud(font_path=FONT_ROOT + 'arial.
    ttf',stopwords=STOPWORDS, background_color='black', width=1200,
    height=1000).generate(' '.join(word_string))
    return wordcloud
```

Ross pointed out that a prerequisite to the code in Listing 4-15 is that a folder by the name of Fonts should be created in the same repository as the script itself, and the Arial font file should be pushed into it. For testing he recommended passing in any string input as the parameter to this function for it to generate a wordcloud.

Ross went on further and wrote the code in Listing 4-16 to generate a wordcloud for each of the nine clusters.

Listing 4-16. Creating Function to Plot Wordcloud for Each Cluster

```
def plot_wordcloud(df, clusters, pivot):

    fig = plt.figure(figsize=(15,29.5))
    for cluster in range(clusters):
        List_ = []

        for x in df[df['cluster']==cluster][pivot]:
            try:
                List_.extend(x.split('\n'))
            except:
                pass

        if List_:
            ax = fig.add_subplot(5,2,cluster+1)
            wordcloud = wordcloud_object(List_)
            plt.title('Cluster: %d'%(cluster+1))
            ax.imshow(wordcloud)
            ax.axis('off')
```

Before moving on, Ross pointed out that one of the parameters in the `plot_wordcloud` method is the pivot parameter. Pivot is the feature from which the bag of words will be taken to make the wordcloud.

Ross was excited as to see what the wordclouds would depict. Hence without waiting any longer he wrote the code in Listing 4-17 to call the method initialized in Listing 4-16, and passed in "keywords" as the pivotal feature.

Listing 4-17. Generating Wordclouds of Feature Named 'Keywords'

```
plot_wordcloud(df, cluster.n_clusters, 'keywords')
```



Figure 4-8. *Wordcloud for each cluster generated from keywords*

After looking at Figure 4-8, Ross decided to give his deductive abilities a shot and defined the nine clusters as follows:

- Cluster 1: Papers talking about search and robotics
- Cluster 2: Papers talking in depth about models' learning and optimization
- Cluster 3: Topics of application of data analytics in games and social media analytics
- Cluster 4: Topics of image recognition, robotics, and social media analytics
- Cluster 5: Topics of linear programming and search
- Cluster 6: Papers on reasoning-based models
- Cluster 7: Papers on application of data sciences in social graphs and other online mediums
- Cluster 8: Topics ranging in knowledge graphs
- Cluster 9: Papers concentrating on game theory and data security

Ross pointed to the fact that the words in the wordclouds are unigram (i.e., single words). He was happy with the results so far, but it seemed to him that some clusters had overlapping terms; for instance, clusters 1 and 5 had "search" as the overlapping term. It felt important to Ross to remove the overlap to make the cluster keywords and characteristics as distinctive as possible. Hence he planned to delve deeply into the details to see if both the clusters touched the same or distinctive topics of SEO. He started off by defining a method in Listing 4-18 which will take the term (e.g., "search") as a parameter and return the keywords for each cluster having that term within.

Listing 4-18. Define Method to Find Complete Keywords for Given Clusters and Unigram

```
def perform_cluster_group_audit(clusters, term):

    for cluster in clusters:

        df_cluster = df[df['cluster'] == cluster]
        print 'Cluster number: %d'%(cluster + 1)
        keywords = list(df_cluster['keywords'])
        keywords = [keyword.split('\n') for keyword in keywords]
        keywords = [item for sublist in keywords for item in sublist]
        keywords = [keyword.lower() for keyword in keywords if term in
        keyword.lower()]
        keywords_freq = {x:keywords.count(x) for x in keywords}
        print sorted(keywords_freq.items(), key=operator.itemgetter(1),
        reverse=True)
        print '\n'
```

Ross pointed out the second to last line of Listing 4-18 which is used to sort the keywords in descending order of their frequencies. Now it was time to use the method defined in Listing 4-18. He started off by looking at the topics that differentiate clusters 1 and 5 in the aspect of "search."

Listing 4-19. Using Function to Find Keywords for Search in clusters 0 and 4

```
perform_cluster_group_audit([0,4], 'search')
```

Output

```
Cluster number: 1
[('heuristic search', 7), ('greedy best first search', 4), ('Monticello
tree search', 2), ('bounded suboptimal search', 1), ('real-time search',
1), ('best-first search', 1), ('incremental search', 1), ('parallel
search', 1), ('similarity search', 1), ('suboptimal heuristic search', 1),
('agent-centered search', 1), ('approximate nearest neighbor search', 1),
('hierarchical search', 1)]

Cluster number: 5
[('search', 3), ('heuristic search', 3), ('local search', 2), ('stochastic
local search', 2), ('and/or search', 2)]
```

Ross made clear that clusters 1 and 5 were referred to as 0 and 4 because list indexes start from 0. The output of Listing 4-19 depicted topics along a heuristic search being captured by both the segments. He noticed that cluster 5 had topics on local search and cluster 1 had topics on other search algorithms.

After looking at Figure 4-8 Ross noticed that clusters 3, 4, and 7 were sharing topics along the social medium space. Hence he decided to investigate the points of differences by using the function earlier defined in Listing 4-20.

Listing 4-20. Using Function to Find Keywords for Social in Clusters 2, 3, and 6

```
perform_cluster_group_audit([2,3,6], 'social')
```

Output

```
Cluster number: 3
[('computational social choice', 11), ('social choice theory', 2), ('social
decision schemes', 2), ('randomized social choice', 1)]

Cluster number: 4
[('social media', 5), ('social spammer', 2), ('social image classification', 2)]

Cluster number: 7
[('social networks', 6), ('social network', 3), ('social infectivity',
3), ('social network analysis', 2), ('location based social network', 2),
('social influence', 2), ('social dynamics', 1), ('social explanation', 1)]
```

The output of Listing 4-20 made it clear to Ross how the three clusters were different from each other. Cluster 3 emphasized social choice theories. Cluster 4, on the other hand, concentrated on social images classification and social spamming. Finally, cluster 3 grouped research papers related to social graphs and influences among interconnections.

Ross was curious to see what the wordclouds would look like if the pivotal feature had been other than ‘keywords.’ Ross had limited time and too much to do, so he invites you to come on board and help him answer the questions in the Exercises.

EXERCISES

1. Plot wordclouds while keeping “groups” as the pivotal feature.
2. Plot wordclouds while keeping “topics” as the pivotal feature.

Ross had read somewhere that clusters in k-means are usually in a spherical shape. He was curious to know if methods exist which create non-uniform clusters of different shapes. It didn’t take him long to discover Gaussian mixture models as a potential candidate to satisfy his curiosity.

Gaussian Mixture Model

Gaussian mixture models use probabilistic theory to estimate the number of clusters the data can be divided into. It works on the assumption that all data points are generated from the mixture of finite number of Gaussian distributions with unknown parameters. Gaussian mixture models can be thought as an enhancement to k-means where the covariance of the data and centers of Gaussian models are taken into consideration.

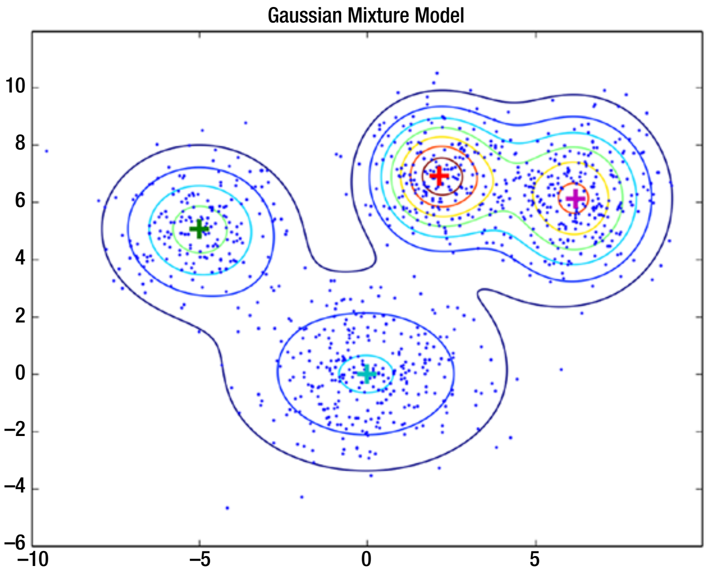


Figure 4-9. *Visual representation of Gaussian mixture model*

Gaussian mixture model comes in the following variants to constrain the covariance of different cluster estimates:

- Spherical
- Diagonal
- Tied
- Full covariance

Ross decided to apply Gaussian mixture models to find out the segments and their characteristics. For that purpose he wrote a function in Listing 4-21 to generate a scatter plot of clusters generated from Gaussian mixture models.

Listing 4-21. Defining Function to Plot the Clusters

```
def plot_results(X, Y_, means, covariances, index, title):

    color_iter = itertools.cycle(['b', 'g', 'red', 'm', 'y', 'navy', 'c',
    'cornflowerblue', 'gold',
                                  'darkorange'])
    splot = plt.subplot(2, 1, 1 + index)
    for i, (mean, covar, color) in enumerate(zip(
            means, covariances, color_iter)):
        v, w = np.linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / np.linalg.norm(w[0])

        if not np.any(Y_ == i):
            continue
        plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], .8, color=color)

        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi  # convert to degrees
        ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle,
        color=color)
        ell.set_clip_box(splot.bbox)
        ell.set_alpha(0.5)
        splot.add_artist(ell)

    plt.xlim(0.0, 0.1)
    plt.ylim(-0.2, 1.2)

    plt.xticks(())
    plt.yticks(())
    plt.title(title)
```

In Listing 4-21 Ross made sure to represent each cluster with a different color in order to help visually differentiate among them. He was now looking forward to applying Gaussian mixture models; however, he wasn't very sure about what the number of

components should be. He looked through the literature and found out that BIC score is used commonly in Gaussian mixture models for deciding upon the number of clusters. Now his plan was to run a Gaussian mixture model for all possible combinations of covariance types and components (i.e., from 2 to 9). Covariance types include spherical, tied, and full. For each of these combinations BIC score will be calculated and the combination with the maximum BIC score will be selected. Ross made this happen by writing the code in Listing 4-22.

Listing 4-22. Determining the Optimal Covariance Type and Components for Model and Plotting It

```
matrix, x_cols = matrix_from_df(data_train)
X  = matrix[x_cols].as_matrix()
model_stats = []
n_components_range = range(2, 10)
cv_types = ['spherical', 'tied', 'full']
for cv_type in cv_types:
    for n_components in n_components_range:

        gmm = mixture.GaussianMixture(n_components=n_components,
                                      covariance_type=cv_type, random_
                                      state=0)
        gmm.fit(X)
        model_stats.append({'name':'%s_%d'%(cv_type, n_components),
        'model':gmm, 'bic':gmm.bic(X)})

bic = np.array([m_type['bic'] for m_type in model_stats])
best_gmm = model_stats[bic.argmax()]
clf = best_gmm['model']
color_iter = itertools.cycle(['navy', 'turquoise', 'cornflowerblue'])

bars = []

# Plot the BIC scores
spl = plt.subplot(2, 1, 1)
for i, (cv_type, color) in enumerate(zip(cv_types, color_iter)):
    xpos = np.array(n_components_range) + .2 * (i - 2)
    bars.append(plt.bar(xpos, bic[i * len(n_components_range):
                                  (i + 1) * len(n_components_range)],
                        width=.2, color=color))
plt.xticks(n_components_range)
plt.ylim([bic.min() * 1.01 - .01 * bic.max(), bic.max()])
plt.title('BIC score per model')
spl.set_xlabel('Number of components')
spl.legend([b[0] for b in bars], cv_types)

labels = clf.predict(X)
plot_results(X, labels, gmm.means_, gmm.covariances_, 1,
             'Gaussian Mixture-%s'%gmm.converged_)
```

```
plt.xticks(())
plt.yticks(())
plt.title('Selected GMM: %s model, %s components'%(best_gmm['name'].
split('_')[0], best_gmm['name'].split('_')[1]))
plt.subplots_adjust(hspace=.35, bottom=.02)
plt.show()
```

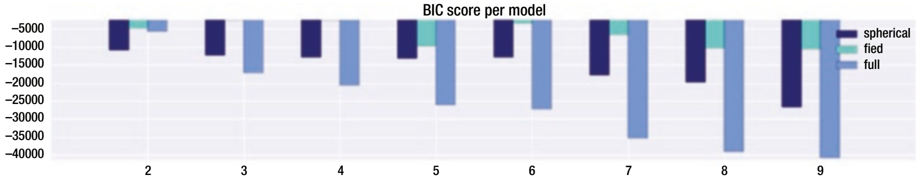


Figure 4-10. *BIC score per model*

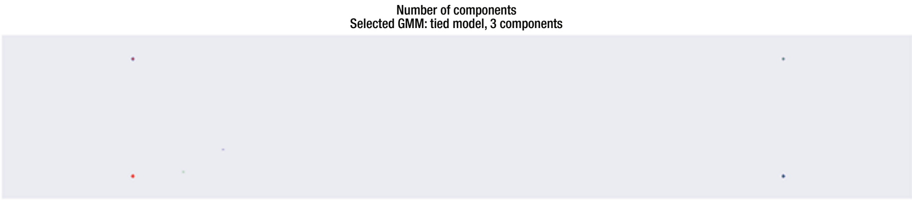


Figure 4-11. *Best Gaussian mixture model's components plot*

Ross pointed out that the values within Figure 4-10 are of a negative magnitude. Therefore, a covariance type and number of components at 0 will be deemed the winner. A Gaussian mixture model with covariance type "tied" and three components qualified to be the winner in this case. Ross was curious to see if all the cluster sizes were uniform or not. For that purpose he wrote the code in Listing 4-23.

Listing 4-23. Display Frequency of Objects in Each Cluster

```
matrix['cluster'] = labels
matrix.cluster.value_counts()
```

Output

```
0      232
1      108
2      56
Name: cluster, dtype: int64
```

Ross was curious to see what the wordclouds from this method would look like, and if they would make any intuitive sense or not. However, before doing so he had to merge the matrix into the original dataset. He did so with the aid of the code in Listing 4-24.

Listing 4-24. Merging Matrix into the Initial Data Frame

```
customer_clusters.columns.name = None
df = data_train.merge(customer_clusters, on='title')
```

Without waiting any longer, he pushed the merged dataset in Listing 4-25 to the function defined in Listing 4-15 for generating the wordcloud plots.

Listing 4-25. Generating Wordclouds of Feature Named "Keywords"

```
plot_wordcloud(df, gmm.n_components, 'keywords')
```

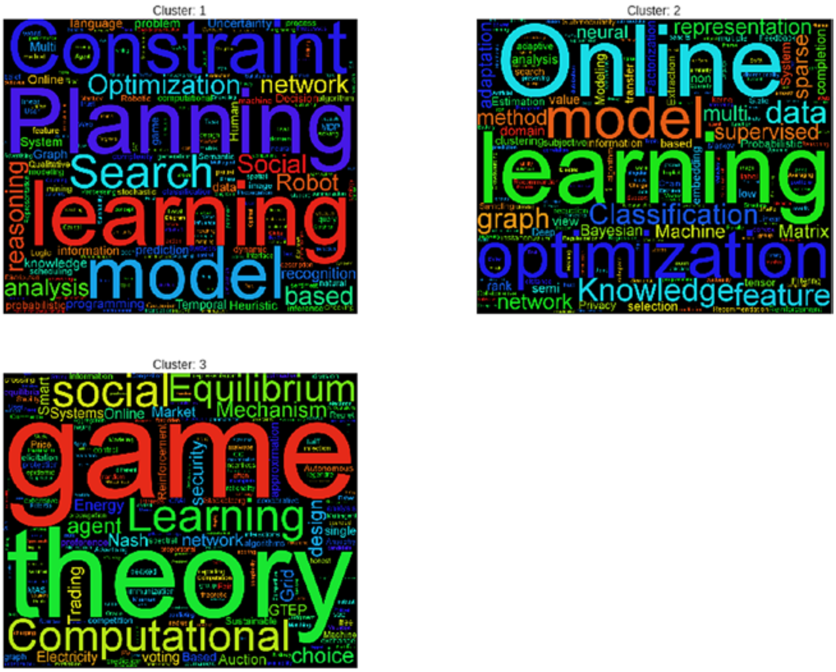


Figure 4-12. *Wordcloud for each cluster generated from keywords*

Ross decided to put his deductive abilities to the test and defined the research papers that fall within each of these clusters:

- Cluster 1: Papers discussing in depth models' learning and linear programming
- Cluster 2: Papers discussing in depth model optimization and knowledge graphs
- Cluster 3: Topics on game theory and social media analytics

Ross was thrilled to see the results, as these clusters seemed to be unique in terms of their characteristics. However, he was searching for an algorithm which can automatically determine the optimal number of components. After some struggle he was able to settle on the Bayesian Gaussian mixture model.

Bayesian Gaussian Mixture Model

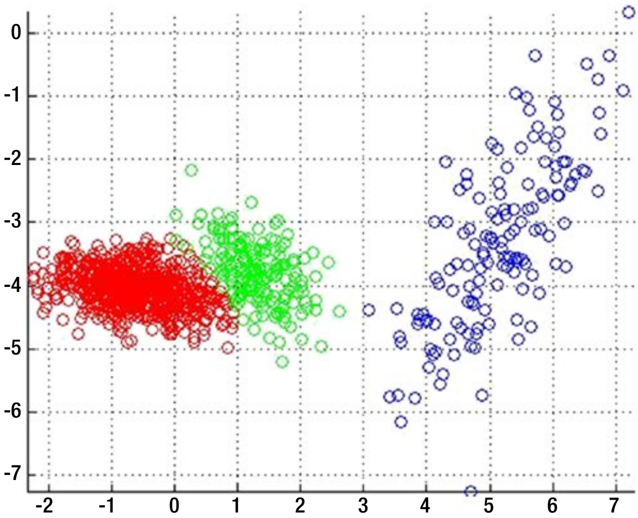


Figure 4-13. *Illustration of Bayesian Gaussian mixture model*

The Bayesian Gaussian mixture model is a variant of the Gaussian mixture model in which the model chooses the optimal number of clusters on its own. It exhibits this behavior when `weight_concentration_prior` is set to be small enough and `n_components` is set to be larger than what is found necessary by the model. The variational Bayesian mixture model sets some mixture weight values close to zero, which lets the model choose effective components automatically.

Ross decided to apply the Bayesian Gaussian mixture model to the data matrix. He decided to limit the number of components to five. He was interested to see what number of components will be deemed optimal by the Bayesian Gaussian mixture model.

Listing 4-26. Training the Model and Plotting the Clusters

```
matrix, x_cols = matrix_from_df(data_train)
X  = matrix[x_cols].as_matrix()

dpgmm = mixture.BayesianGaussianMixture(n_components=3,
                                        covariance_type='full', random_
                                        state=1).fit(X)
```

```
labels = dpgmm.predict(X)
plot_results(X, labels, dpgmm.means_, dpgmm.covariances_, 1,
             'Bayesian Gaussian Mixture with a Dirichlet process
             prior-%s'%dpgmm.converged_)

plt.show()
```

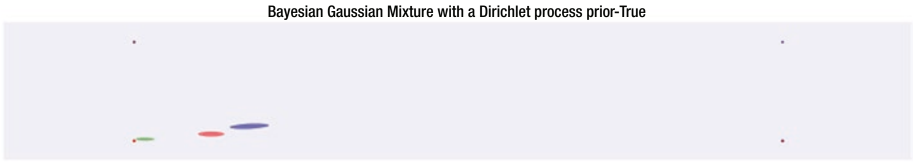


Figure 4-14. *Clusters formed by means of Bayesian Gaussian mixture model*

The three colors in Figure 4-14 implied to Ross that the three components were sufficient for the model to converge. He was interested to see the frequency of research papers that fall within each of these clusters; hence, he wrote the code snippet in Listing 4-27.

Listing 4-27. Display Frequency of Objects in Each Cluster

```
matrix['cluster'] = labels
matrix.cluster.value_counts()
```

Output

```
0      229
1      117
2      50
Name: cluster, dtype: int64
```

Ross noticed that the first cluster has more than 50% of the observations falling within itself, with the number of observations decreasing in the next two clusters.

The current matrix only had titles and clusters as the features. Hence he decided to merge them within the initial data frame in order to get access to other features corresponding to these research paper titles.

Listing 4-28. Merging Matrix into the Initial Data Frame

```
customer_clusters.columns.name = None
df = data_train.merge(customer_clusters, on='title')
```

It was time for Ross to explore the characteristics of each of those three clusters. Hence he brought the function defined in Listing 4-16 into application.

Listing 4-29. Generating Wordclouds of Feature Named "Keywords"

```
plot_wordcloud(df, dpgmm.n_components, 'keywords')
```

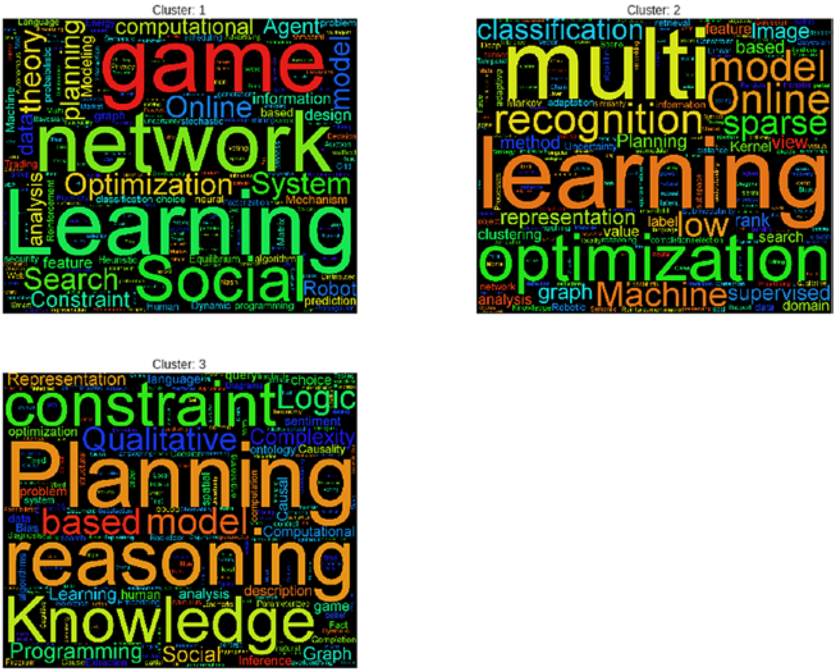


Figure 4-15. *Wordcloud for each cluster generated from keywords*

Ross defined the three clusters with the aid of the wordclouds in the following words:

- Cluster 1: Papers discussing in depth game theory and social media analytics
- Cluster 2: Papers discussing in depth model optimization and models' learning
- Cluster 3: Topics on linear programming, knowledge graphs, and reasoning-based models

Ross was satisfied as after going through the exercise he was able to get finite clusters with distinct characteristics. He planned to drop the output of k-means and go forward with the segments defined by the Bayesian Gaussian mixture model. His reasoning was, first, that the clusters made more intuitive sense, and second, that it was smart enough to find the optimal number of clusters all on its own.

Though Ross had nailed the problem at hand, he was interested to see the applications of clustering in the real world.

Applications of Clustering

Applications of clustering are vibrant in several fields of study.

Identifying Diseases

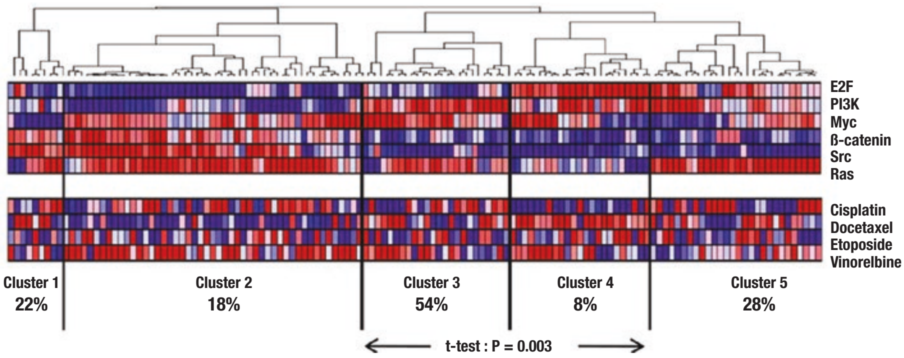


Figure 4-16. *Clusters obtained along with their confidence intervals while identifying diseases*

Clustering has recently caught the attention of those in the medical field. This is primarily due to the fact that unsupervised non-linear clustering algorithms have been relatively more effective in predicting cancerous diseases.

Document Clustering in Search Engines

Search engines use the input query of user as a parameter to cluster documents from the information retrieval system. The search results are thus in a sorted order of those clustered documents.

Demographic-Based Customer Segmentation

Clustering is also used in the field of marketing—more specifically, branding. Marketing research executives use clustering to find out customer segmentations and perceptual maps of their brand and that of their competitors. This way they find out the positioning of their brands and the customer segmentation they are catering to. This helps brands determine the customer personas to target, both existing and potential customers, and their demographics.

Ross decided to conclude his journey by recalling how he had started off with the difference between supervised and unsupervised algorithms. He then transformed the data into a matrix to increase the dimensions of the data at hand. He looked at different methodologies for determining the optimal number of clusters. He learned how to make wordclouds to generate an intuitive sense of the clusters formed. Along the lines of the end objective, he used several techniques in search of the right one to divide the research papers into segments with disjoint characteristics. His plan was to market those segments by means of their related keywords for the upcoming conference this year. He selected the three components from the Bayesian Gaussian mixture model to fuel up his efforts for the upcoming conference next year.

CHAPTER 5

Classification

This chapter aims to solve real-world problems that depend on a finite number of outcomes. These outcomes can be Boolean in nature, with only two choices (i.e., True/False or Yes/No). They can also be nominal in nature (boat name on which a passenger will embark, maximum education attainment a group of students will have, etc.). Throughout this chapter we will be talking about supervised learning whereby labeled data will be used to train the model. Training the models on this data will enable label predictions on an unseen dataset (i.e., future predictions). In this chapter we will be talking about feature extraction and selection and will conclude with methods to evaluate classification accuracy.

Note This book incorporates **Python 2.7.11** as the de facto standard for coding examples. Moreover, you are required to install it for the *Exercises*.

In this chapter we will be incorporating the **Medical Appointment No-Shows dataset** available to download from

`www.kaggle.com/joniarroba/noshowappointments`

Case Study: Ohio Clinic—Meeting Supply and Demand

Dr. Judy, a pediatric surgeon and clinic supervisor at Ohio Clinic, was in big trouble, facing clinic losses for the third consecutive year. Dr. Judy had recently been promoted to this position, but she knew for a fact that the clinic had been doing due diligence in terms of efficiency. What surprised her most was that the hospital was incurring losses despite having the finest doctors available and no lack of scheduled appointments. To be reassured about the financial side of things, she hired a third-party firm to audit the finance department. However, the firm found no evidence pertaining to the dilemma at hand.

Ohio Clinic is a nonprofit medical center in Ohio. The clinic operates with a unique mission of blending research and education with clinical and hospital care. The medical center has a huge head force of 50,000 employees, and as a result of the combined effort of those employees, the medical center has been able to handle approximately 7 million visits so far.

D. Haroon, *Python Machine Learning Case Studies*, DOI 10.1007/978-1-4842-2823-4_5

Previously, in situations like these, the hospital relied on an influx of capital from donors. However, with the recent change in management, efforts had been commenced to make the medical center self-sustainable. As Dr. Judy recalled,

> *This new initiative by the management was a huge challenge for me given the fact that it was the exact time when I was handed over the position. I got no time to settle and thus it was a do-or-die situation for me. If finances have no flaws, appointments are skyrocketing, no considerable hiring, or raises have taken place, then the only reason for this dilemma could be that patients are not showing up after getting the appointments in the first place.*

A board meeting was expected soon and she had to gather insights by then to give a reason to this anomaly. Dr. Judy knew that neither speculations nor rumors would enable her to figure out the reason for the losses despite the number of scheduled appointments. To further investigate her suspicions, she obtained the data dump of appointments, and decided to use the data to validate the evidence. She knew that concrete data is not subject to bias and thus would provide a clear picture of the situation at hand. First she had to cross-check whether the features within the dataset were relevant to the problem at hand. Hence she fetched the data dictionary in Table 5-1.

Table 5-1. *Data Dictionary for the Medical Appointment No-Shows Dataset*

Feature name	Description
Age	Age of patient
Gender	Gender of patient
AppointmentRegistration	Date on which appointment was issued to the patient
ApointmentData	Date for which appointment was issued to the patient
DayOfTheWeek	Day of the week for which appointment was issued
Status	Day of the week for which appointment was issued (i.e., response variable)
Diabetes	Whether the patient has diabetes or not
Alcoolism	Whether the patient is affected by Alcoolism or not
HiperTension	Whether the patient has HiperTension or not
Handicap	Whether the patient is handicapped or not
Smokes	Whether the patient smokes or not
Tuberculosis	Whether the patient has tuberculosis or not
Scholarship	Whether or not a patient has been granted scholarship from a social welfare organization or not. Poor families may benefit by receiving financial aid.
Sms_Reminder	Whether SMS reminder for appointment has been issued to the patient or not
AwaitingTime	AwaitingTime = AppointmentRegistration – ApointmentData

Dr. Judy searched the Web to find all the relevant packages she would need throughout the course of this analysis. Listing 5-1 illustrates the results of her Web search.

Listing 5-1. Importing Packages Required for This Chapter

```
%matplotlib inline

import numpy as np
import pandas as pd
from time import time
import matplotlib.pyplot as plt
from IPython.display import Image
from matplotlib.pylab import rcParams

from sklearn import metrics
from sklearn.cross_validation import train_test_split

from sklearn.decomposition import PCA
from sklearn import kernel_approximation
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.kernel_approximation import (RBFSampler,Nystroem)
from sklearn.ensemble import GradientBoostingClassifier,
RandomForestClassifier

rcParams['figure.figsize'] = 15, 5
```

Dr. Judy was hoping to do the following with the information from the data dump:

1. Discover reasons that losses are coming up even though the rate of appointments is going up?
2. If patients are not reporting at the time of their scheduled appointments, come up with a method to determine whether a patient would show up on the basis of his/her characteristics. She believed that knowing which patients were likely not to show up would enable the hospital to take countermeasures like the following:
    - Provide constant appointment reminders and confirmations
    - Make the head count of doctors and hospital staff in line with the demand at hand.

Dr. Judy decided to start off by understanding the features within the data dump and listing their possible values.

Features' Exploration

Dr. Judy put her faith in the data dump and loaded it into the computer's memory using the code snippet in Listing 5-2. She also made sure to print the first initial observations from the data dump.

Listing 5-2. Reading the Data in the Memory and Printing the First Few Observations

```
data = pd.read_csv('examples/No-show-Issue-Comma-300k.csv')
data.head()
```

Table 5-2. *Print of Observations of the Dataset*

	Age	Gender	AppointmentRegistration	AppointmentData	DayOfTheWeek	Status	Diabetes	Alcoolism	HiperTension	Handcap	Smokes	Scholarship	Tuberculosis	Sms_Reminder	AwaitingTime
0	19	M	2014-12-16T14:46:25Z	2015-01-14T00:00:00Z	Wednesday	Show-Up	0	0	0	0	0	0	0	0	-29
1	24	F	2015-08-18T07:01:26Z	2015-08-19T00:00:00Z	Wednesday	Show-Up	0	0	0	0	0	0	0	0	-1
2	4	F	2014-02-17T12:53:46Z	2014-02-18T00:00:00Z	Tuesday	Show-Up	0	0	0	0	0	0	0	0	-1
3	5	M	2014-07-23T17:02:11Z	2014-08-07T00:00:00Z	Thursday	Show-Up	0	0	0	0	0	0	0	1	-15
4	38	M	2015-10-21T15:20:09Z	2015-10-27T00:00:00Z	Tuesday	Show-Up	0	0	0	0	0	0	0	1	-6

Dr. Judy decided to use Table 5-2 as an aid to classify the features into the most common data types.

- **Integer:** Age, waiting time
- **String:** Gender, DayOfTheWeek, Status
- **Datetime:** AppointmentRegistration, ApointmentData
- **Boolean:** HiperTension, Handicap, Smokes, Scholarship, Tuberculosis, Sms_Reminder

She anticipated the records within the data dump to be in thousands as she noticed that the appointment calendars were usually full at the start of every month. She decided to validate that in Listing 5-3.

Listing 5-3. Finding Count of Number of Observations in the Dataset

```
print len(data)
```

Output

```
300000
```

The output of Listing 5-3 was way above Dr. Judy's expectations and reassured her of the fact that insights gained from the data will have a high confidence level. Next she planned on observing the number of distinct values within each of the features. She believed this would help her to drop the features that offered no variability (i.e., having same value for all observations) and to decide upon the types of plots these features could be represented in. For that purpose she wrote the code in Listing 5-4.

Listing 5-4. Print of the Frequency of Distinct Values in Each Feature Set

```
for column in list(data.columns):
    print "{0:25} {1}".format(column, data[column].nunique())
```

Output

```
Age                             109
Gender                          2
AppointmentRegistration         295425
ApointmentData                  534
DayOfTheWeek                    7
Status                          2
Diabetes                        2
Alcoolism                       2
HiperTension                    2
Handicap                        5
Smokes                          2
Scholarship                     2
Tuberculosis                    2
Sms_Reminder                    3
AwaitingTime                    213
```

Dr. Judy explained that {0:25} in the print statement meant that the feature in the first index (i.e., column) would be printed and 25 character spaces would be allocated to it. The output of Listing 5-4 implied that there are many variables within the dataset that have discrete values, examples of which include Gender, Status, DayOfTheWeek, and so on. This helped her choose the types of plots to represent each of these features. She decided to plot AwaitingTime and Age as histograms because these are numbers and continuous in nature. As for the features which are discrete in nature, she planned to represent them as a bar graph. She executed that graph by initializing the function in Listing 5-5, and later called it in Figure 5-1 to plot the figure.

Listing 5-5. Initialize Function to Plot All Features Within the Dataset

```
def features_plots(discrete_vars):

    plt.figure(figsize=(15,24.5))

    for i, cv in enumerate(['Age', 'AwaitingTime']):
        plt.subplot(7, 2, i+1)
        plt.hist(data[cv], bins=len(data[cv].unique()))
        plt.title(cv)
        plt.ylabel('Frequency')

    for i, dv in enumerate(discrete_vars):
        plt.subplot(7, 2, i+3)
        data[dv].value_counts().plot(kind='bar', title=dv)
        plt.ylabel('Frequency')
```

Dr. Judy pointed to the fact that the method in Listing 5-5 takes as an input the names of features which are discrete in nature because those will be represented as bar graphs. Without wasting time she called the function to plot the feature representations in Listing 5-6.

Listing 5-6. Calling 'features_plot' method in Listing 5-5 to Plot the Feature Representation

```
discrete_vars = ['Gender', 'DayOfTheWeek', 'Status', 'Diabetes',
                     'Alcoolism', 'HiperTension', 'Handcap', 'Smokes',
                         'Scholarship', 'Tuberculosis', 'Sms_Reminder']

features_plots(discrete_vars)
```

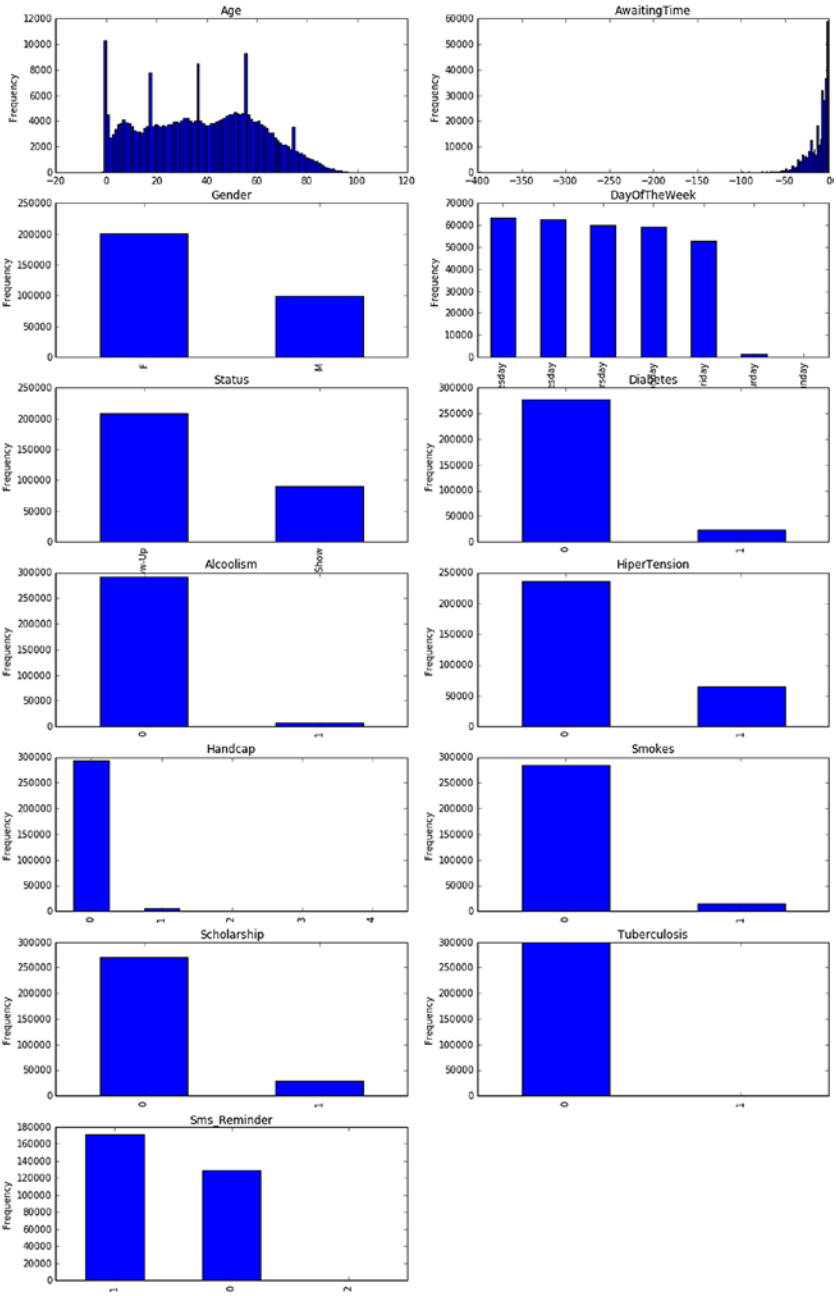


Figure 5-1. *Plots of feature representations*

While looking at Figure 5-1, Dr. Judy noticed that none of the features has missing values. She made the following deductions about some of the features from the dataset:

- **Age:** Age lay in the range of -2 and 113. Age between 0 and 113 did make sense, but what surprised her was how it could be negative. It seemed to her that there was some noise in the dataset.
- **Handicap:** Instead of being Boolean, this feature had values in the range of 0 and 4.
- **Sms_Reminder:** Instead of being a Boolean entity, it had values in the range of 0 and 2. It seemed to her that Sms_Reminder represented the frequency of reminders sent to each and every patient.
- **AwaitingTime:** Dr. Judy was puzzled to see AwaitingTime in negative terms. By definition this feature represented the number of days from which the appointment was issued to the date for which the appointment was issued. She believed that positive numbers would have made more sense.

Findings from the deductions meant that Dr.Judy would have to clean data a bit (i.e., perform data wrangling) before pushing it into the analysis pipeline.

Performing Data Wrangling

She read somewhere that data wrangling is the process of mapping raw data into an organized form suitable for analyzing the data. Findings from Figure 5-1 depicted the presence of negative values within 'Age,' which didn't make any intuitive sense to her, and thus she referred to these as noise. Hence she planned to start the treatment by counting the frequency of negative Age values. If the frequency of these observations were high she wouldn't delete them as doing so would keep her from detecting the variations necessary for the prediction. To the contrary she planned to delete them if their frequency was negligible as deleting them wouldn't have a major effect on the analysis. She started off by taking the frequency of negative Age values in Listing 5-7.

Listing 5-7. Counting Frequency of Negative Age Observations

```
data[data['Age'] < 0]['Age'].value_counts().sum()
```

Output

```
6
```

The output of Listing 5-7 indicated the negative Age observations to be negligible as compared to the total number of observations within the dataset. Hence Dr. Judy decided to remove these negative observations from the Age feature.

Listing 5-8. Removing Observations with Negative Age Values

```
data = data[data['Age'] >= 0]
```

While looking at the deductions from Figure 5-1, Dr. Judy also noticed that the feature named 'Handicap' was an integer variable rather than a Boolean variable one. She recalled that the data definition for this feature set was missing from the data source and hence open to multiple interpretations. The only rationale she could think of for values lying within the range of 0 and 4 was that this might be a Likert scale question or a numeric entity representing the frequency of times a patient has been handicapped. As the interpretation of this feature was ambiguous to her she decided to drop this feature from the dataset.

Listing 5-9. Removing Variable Named 'Handicap' from the Dataset

```
del data['Handcap']
```

While looking at the deductions from Figure 5-1, Dr. Judy also recalled that values within AwaitingTime appeared to be negative, and hence it made sense to transform them into positive values.

Listing 5-10. Making Values Within AwaitingTime Positive

```
data['AwaitingTime'] = data['AwaitingTime'].apply(lambda x: abs(x))
```

The treatment of features on the basis of her deductions in Figure 5-1 was completed. However, Dr. Judy recalled reading that most machine learning algorithms work best with integer or floating point input rather than in string format. She knew that the categorical variables in the dataset didn't comply with this condition and hence it was important for her to recode the string categorical features to their integer counterparts. Dr. Judy started off by recoding DayOfTheWeek by binding the mapping from string to integer.

Listing 5-11. Recode String Categorical Feature DayOfTheWeek to Integers

```
dow_mapping = {'Monday' : 0, 'Tuesday' : 1, 'Wednesday' : 2, 'Thursday' : 3,
'Friday' : 4, 'Saturday' : 5, 'Sunday' : 6}
data['DayOfTheWeek'] = data['DayOfTheWeek'].map(dow_mapping)
```

Dr. Judy explained her recoding methodology as follows:

> *There exists methods for automatic recoding of categorical variables. These methods perform the recoding in an alphabetic order such that if categories for a feature are Male and Female, then it will assign a code of 0 to Female and 1 to Male. DayOfTheWeek is an ordinal quantity; hence I decided to do the mapping manually. However, this is not the case in the Gender and Status features as they are nominal in nature. Hence, I will be using the predefined method to do the recoding automatically.*

Dr. Judy performed the recoding for 'Gender' and 'Status' in Listing 5-12.

Listing 5-12. Recode String Categorical Features to Integers

```
for field in ['Gender', 'Status']:
    data[field] = pd.Categorical.from_array(data[field]).codes
```

Once she had performed data wrangling on the original dataset, Dr. Judy was curious as to how their representations had changed. Hence she looked at Figure 5-2 to see their transformed feature representations.

Listing 5-13. Feature Representations Post Data Wrangling

```
discrete_vars = ['Gender', 'DayOfTheWeek', 'Status', 'Diabetes',
                     'Alcoolism', 'HiperTension', 'Smokes',
                         'Scholarship', 'Tuberculosis', 'Sms_Reminder']

features_plots(discrete_vars)
```

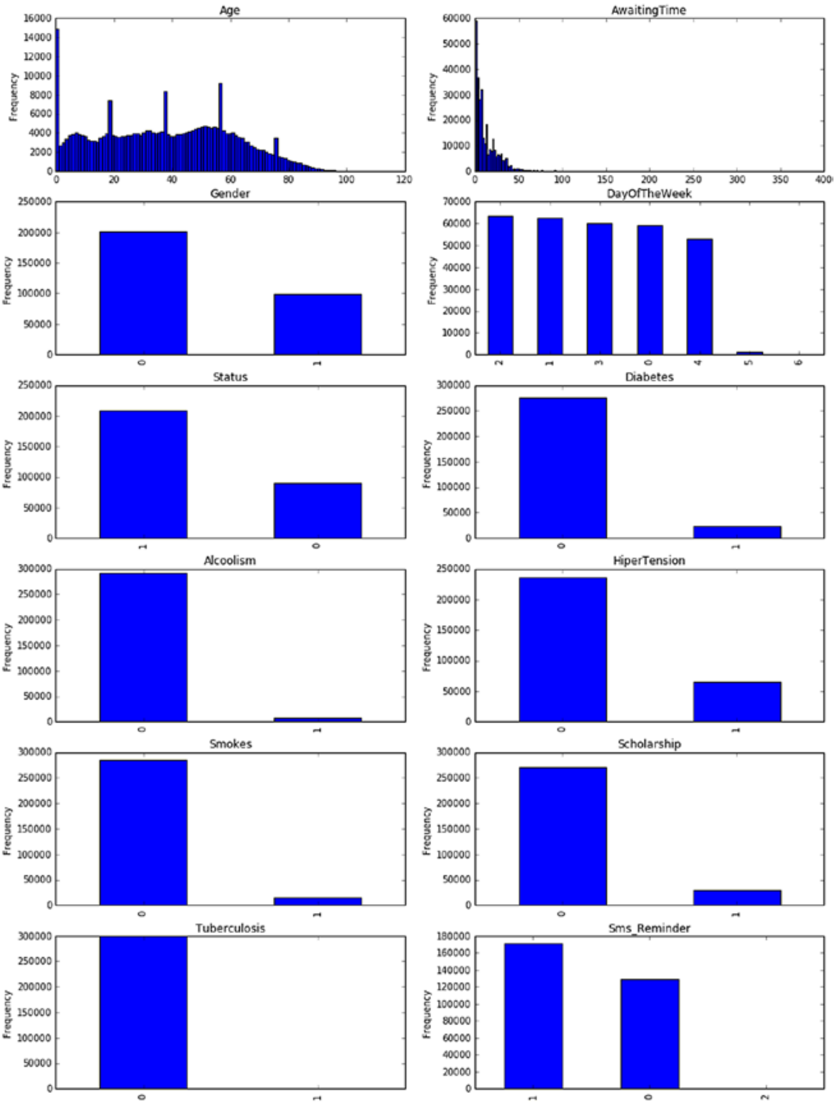


Figure 5-2. *Plots of feature representations post data wrangling*

Dr. Judy made sure to remove 'Handicap' from the list of discrete features in Listing 5-13. She pointed out that recoding assigned codes in the range of

0 to (number of unique observations within the feature - 1)

While looking at Figure 5-2, Dr. Judy noticed that AwaitingTime seemed to decay in an exponential fashion. As per her observation, majority of the patients have an age of 0 (i.e., infants whose age is in months). She also pointed out to the hikes at the ages of 19, 38, and 57. Other than this, another surprising fact was that one-third of patients were males, and that the same proportion of patients didn't show up at the date and time of their appointments. This information gave her a clue as to why the clinic was seeing losses despite of an increase in the number of appointments. She also noticed that majority of the patients were sent at least one SMS reminder; however, two-thirds of the time no reminder was sent. The absence of appointment reminders, she believed might be the reason behind patients not showing up.

Once she understood the features within the dataset and after she had removed the ambiguities by performing data wrangling, Dr. Judy was interested in identifying relationships between different features within the dataset. She wanted to perform this multivariate analysis to gain an intuitive understanding of the types of patients who don't show up on their appointment dates and time.

Performing Exploratory Data Analysis

Rather than starting off with the exploratory data analysis (EDA), Dr. Judy believed it best to see the data visually to identify if there were patterns to which she could redirect the model. Dr. Judy believed that as people got older, their need to see a doctor would also increase. The notion behind this was that as people age, diseases increase exponentially. Hence she wrote the code in Listing 5-14 to see if the relationship between these quantities is inversely proportional in reality or not.

Listing 5-14. Scatter Plot Between Age and AwaitingTime

```
plt.scatter(data['Age'], data['AwaitingTime'], s=0.5)
plt.title('Scatter plot of Age and Awaiting Time')
plt.xlabel('Age')
plt.ylabel('Awaiting Time')
plt.xlim(0, 120)
plt.ylim(0, 120)
```

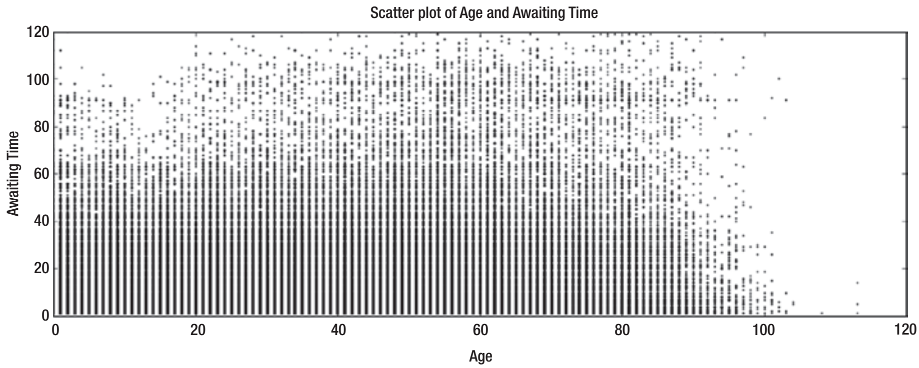


Figure 5-3. *Scatter plot between Age and AwaitingTime*

Dr. Judy wasn't happy with the results as Figure 5-3 gave a highly dispersed plot, with no signs of correlation at the start and a bit of negative correlation after the age of 90. Visual representation showed the earlier established hypothesis to have failed. However, she decided to validate this using a statistical correlation technique as well in Listing 5-15.

Listing 5-15. Calculating Pearson Correlation Between Age and AwaitingTime

```
pd.set_option('display.width', 100)
pd.set_option('precision', 3)
correlations = data[['Age', 'AwaitingTime']].corr(method='pearson')
print(correlations)
```

Output

```
               Age          AwaitingTime
Age            1.00e+00     -4.18e-03
AwaitingTime   -4.18e-03    1.00e+00
```

The correlation between Age and AwaitingTime was approaching 0 which deferred to the hypothesis she had established earlier. She was now interested to see if the increase in SMS reminders increased the likelihood of a patient showing up or not. For that reason she wrote the code in Listing 5-16 to plot a stacked bar graph in Figure 5-4.

Listing 5-16. Effect on Status on the Basis of Number of SMS Reminders

```
data_dow_status = data.groupby(['Sms_Reminder', 'Status'])['Sms_Reminder'].
count().unstack('Status').fillna(0)
data_dow_status[[0, 1]].plot(kind='bar', stacked=True)
plt.title('Frequency of people showing up and not showing up by number of
SMS reminders sent')
plt.xlabel('Number of SMS reminders')
plt.ylabel('Frequency')
```

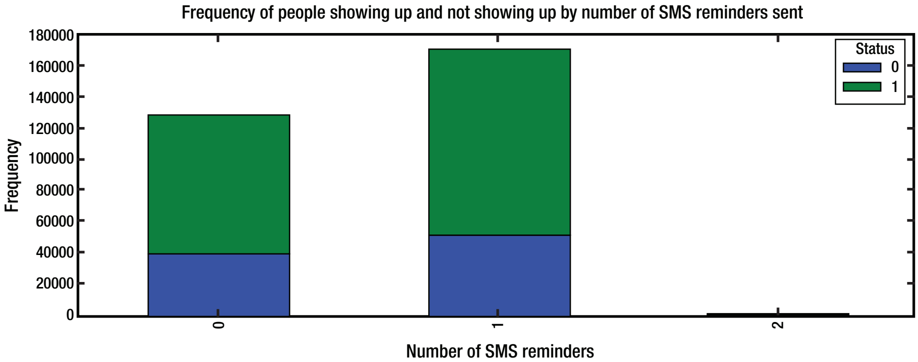


Figure 5-4. *Stacked bar chart between Status and Sms_Reminder*

Looking at Figure 5-4 Dr. Judy noticed that the rate of change in the number of patients who showed up after one reminder relative to those who showed up when no reminder was sent was roughly 30% (i.e., (0.17 m – 0.13 m) / 0/13m), whereas the rate of change in the number of patients not showing up after one reminder relative to those who showed up when no reminder was sent was roughly 25% (i.e., (0.05 m – 0.04 m) / 0/04m0. She thus concluded that SMS reminders do marginally increase the likelihood of a patient showing up on his/her appointment day.

Dr. Judy's perspective was that the majority of the patients would show up in the middle of a week, whereas many of them changed their plans and would not show up for an appointment at the start or the end of the week. This she believed was due to the fact that the start and the end of the week are usually the busiest for businesses, while relief in work occurs during the middle of a week. She decided to validate her perspective in Listing 5-17.

Listing 5-17. Effect on Appointment Day of the Week on the Basis of Number of SMS Reminders

```
data_dow_status = data.groupby(['DayOfTheWeek', 'Status'])['DayOfTheWeek'].
count().unstack('Status').fillna(0)
data_dow_status[[0, 1]].plot(kind='bar', stacked=True)
plt.title('Frequency of people showing up and not showing up by Day of the
week')
plt.xlabel('Day of the week')
plt.ylabel('Frequency')
```

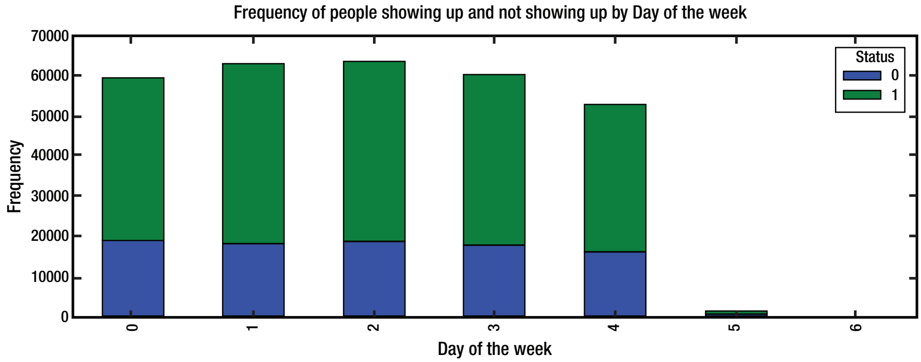


Figure 5-5. *Stacked bar chart between Status and DayOfTheWeek*

Dr. Judy pointed out that in Figure 5-5, 0 represents Monday whereas 6 represents Sunday, with rest of the days lying within in order. Close to no patients showed up on Saturdays, and the clinic was closed on Sundays. The rate of change of no-shows stayed pretty much the same for the remaining five days, whereas the distribution of show-ups seemed to have followed a bell-shaped curve, with most patients showing up on Wednesdays.

Dr. Judy believed that people who are younger are more likely to be busy and to have a job. Because of that they might be short of time and highly likely to not show up at the time of their appointment. She initially thought of finding the measure of center (i.e., mean or median) to validate her hypothesis. However, she wasn't sure which one of the center of measures she should go forward with (i.e., mean, median, or mode).

She recalled from Figure 5-2 that a huge chunk of observations in Age have a value of 0. This she knew could turn out into a disadvantage while calculating the mean as then it will pull the mean toward the lower side. Hence she believed a better measure to be the median, which is unaffected by outliers. Thus she planned on plotting a box plot of Age grouped by Status.

Listing 5-18. Plotting Box Plot of Patients' Age by Status

```
data.boxplot(column=['Age'], return_type='axes', by='Status')
plt.show()
```

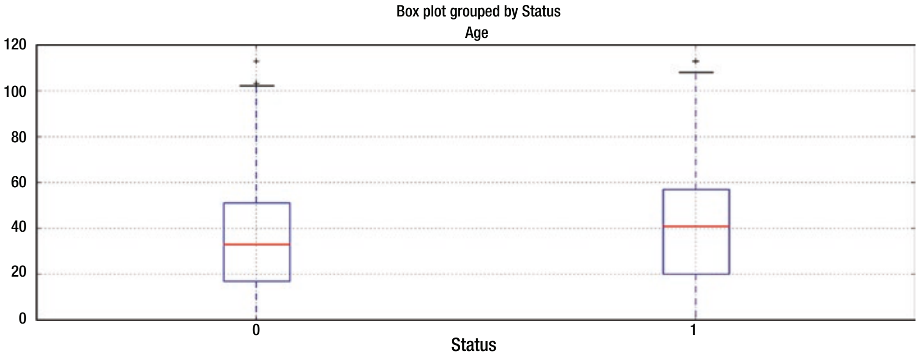


Figure 5-6. *Box plot of Age by Status*

She was thrilled to see the results in Figure 5-6 as her hypothesis held true as the median age of people showing up (i.e., 40) turned out to be relatively greater than that of those not showing up (i.e., 32). She also pointed to the fact that the upper limit (i.e., quartile 3) of people showing up is 58, which is relatively higher than that of those of not showing up (i.e., 52).

Dr. Judy decided to analyze Age against Status for both genders separately. For that reason she wrote the code snippet to plot line graphs in Listing 5-19.

Listing 5-19. Plotting Line Plot of Age by Gender for Patients Status-Wise

```
plt.figure(figsize=(15,3.5))

for i, status in enumerate(['no show ups', 'show ups']):

    data_show = data[data['Status']==i]
    plt.subplot(1, 2, i+1)

    for gender in [0, 1]:
        data_gender = data_show[data_show['Gender']==gender]
        freq_age = data_gender['Age'].value_counts().sort_index()
        freq_age.plot()

    plt.title('Age wise frequency of patient %s for both genders'%status)
    plt.xlabel('Age')
    plt.ylabel('Frequency')
    plt.legend(['Female', 'Male'], loc='upper left')
```

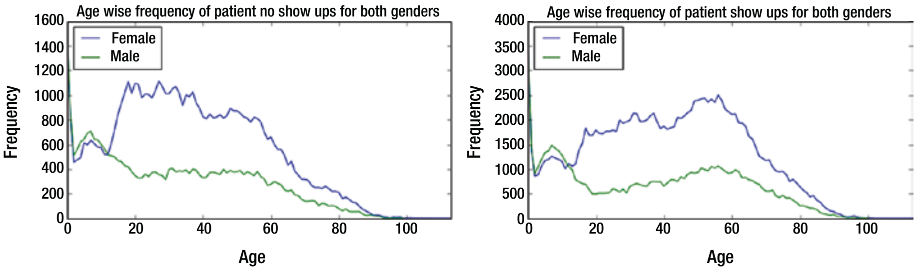


Figure 5-7. *Line chart of Age by Gender status-wise*

Age pattern for males seems to be similar for both statuses, contrary to that of females which varies across statuses. Females in the age range of 42 to 62 are likely to show up on the date and time of their appointment.

Dr. Judy believed that people having a long AwaitingTime (i.e., days to appointment) would have preferred to see another doctor rather than waiting so long. She believed that another reason for not showing up on the appointment date could be that if a disease is seasonal it is likely to be cured after some prevention techniques or home-based remedies. Hence, by the time their appointment date arrives, the patients decide not to consult the doctor. She decided to see if this was true from the data at hand.

Listing 5-20. Plotting Box Plot of AwaitingTime by Status

```
data.boxplot(column=['AwaitingTime'], return_type='axes', by='Status')
plt.show()
```

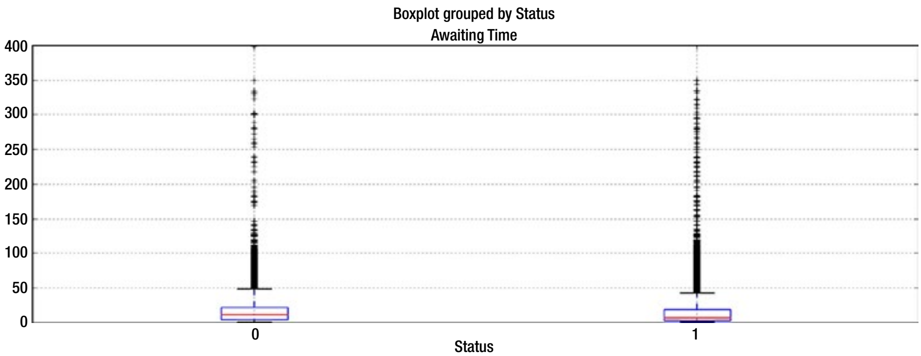


Figure 5-8. *Box plot of AwaitingTime by Status*

After looking at Figure 5-8, Dr. Judy deduced that patients who have their AwaitingTime in the third quartile are relatively more likely to not to show up, which supported her earlier laid null hypothesis—with increasing waiting time the likelihood of patient not showing up also increases. She also pointed out that the median in both the status instances is close to zero.

Dr. Judy was thrilled as exploratory data analysis provided her with many insights. Summing up, she observed that SMS reminders increased the likelihood of a patient showing up on time. She also noticed that younger patients more often did not show up at their appointment times whereas older patients were more disciplined and showed up on time. She also saw that people with long waiting times decided not to show up on their appointed date and times. Dr. Judy now had a fair idea that no-shows to appointment was the reason for losses, and she had gained insight into what leads people to exhibit this behavior. Now she was planning to move ahead and make a classification model to predict the likelihood of a patient to show up or not in the future. This was important because if she knew the total number of patients who would show up on a given day, week, or month, she could tune the resources likewise to save costs. However, before starting with this classification, she wondered whether she could extract more features out of the existing ones by means of features' generation. She believed this would help her capture the variability and definite patterns within the dataset.

Features' Generation

In machine learning, the greater the number of observations and feature sets within the dataset, the greater the likelihood that the model will capture the variability within it, to understand its true essence. Dr. Judy knew that increasing the number of observations was not an option; however, she could increase the feature sets within the dataset. Dr. Judy was clueless as to how she could extract more features from the existing features. After much thinking, she decided to break the features having dates into more granular date components. She wrote the code snippet in Listing 5-21 to apply her logic to features named AppointmentRegistration and AppointmentDate.

Listing 5-21. Breaking Date Features into Date Components

```
for col in ['AppointmentRegistration', 'ApointmentData']:
    for index, component in enumerate(['year', 'month', 'day']):
        data['%s_%s'%(col, component)] = data[col].apply(lambda x:
        int(x.split('T')[0].split('-')[index]))
```

Dr. Judy explained that the code snippet in Listing 5-21 broke AppointmentRegistration and AppointmentData features into their respective year, month, and day components. However, it didn't break them into their time components because ApointmentData doesn't have a time component within its string object, whereas AppointmentRegistration feature does have it. Hence she wrote the code in Listing 5-22 to break the AppointmentRegistration feature into its time component as well.

Listing 5-22. Breaking AppointmentRegistration into Time Components

```
for index, component in enumerate(['hour', 'min', 'sec']):
    data['%s_%s'%('AppointmentRegistration', component)] =
    data['AppointmentRegistration'].apply(
       lambda x: int(x.split('T')[1][:-1].split(':')[index]))
```

Next she wrote the code in Listing 5-23 to witness the step of features' generation.

Listing 5-23. Printing First Few Observations of the Features Extracted Dataset

```
data.head()
```

Table 5-3. *Print of Observations of the Dataset*

	Age	Gender	AppointmentRegistration	AppointmentData	DayOfTheWeek	Status	Diabetes	Alcoolism	HiperTension	Handcap	Smokes	Scholarship	Tuberculosis	Sms_Reminder	AwaitingTime
0	19	M	2014-12-16T14:46:25Z	2015-01-14T00:00:00Z	Wednesday	Show-Up	0	0	0	0	0	0	0	0	-29
1	24	F	2015-08-18T07:01:26Z	2015-08-19T00:00:00Z	Wednesday	Show-Up	0	0	0	0	0	0	0	0	-1
2	4	F	2014-02-17T12:53:46Z	2014-02-18T00:00:00Z	Tuesday	Show-Up	0	0	0	0	0	0	0	0	-1
3	5	M	2014-07-23T17:02:11Z	2014-08-07T00:00:00Z	Thursday	Show-Up	0	0	0	0	0	0	0	1	-15
4	38	M	2015-10-21T15:20:09Z	2015-10-27T00:00:00Z	Tuesday	Show-Up	0	0	0	0	0	0	0	1	-6

Smokes	...	Waiting Time	AppointmentRegistration_year	AppointmentRegistration_month	AppointmentRegistration_day
0	...	29	2014	12	16
0	...	1	2015	8	18
0	...	1	2014	2	17
0	...	15	2014	7	23
0	...	6	2015	10	21

AppointmentData_month	AppointmentData_day	AppointmentRegistration_hour	AppointmentRegistration_min	AppointmentRegistration_sec
1	14	14	46	25
8	19	7	1	26
2	18	12	53	46
8	7	17	2	11
10	27	15	20	9

Looking at Table 5-3 Dr. Judy was thrilled to see the features' generation come true. The evidence was the columns representing the year, month, day, hour, minute, and second components of AppointmentData and AppointmentRegistration. Dr. Judy was well aware that her methodology of increasing the feature sets wasn't the de facto technique as several other techniques achieving the same objective exist. She had some ideas about extracting more features; however, she went with the simplest one to save time. But she is open to collaboration if someone can help her extract other date components from the date features or help her transform the features into their Boolean counterparts. Hence she listed the exercises that follow.

EXERCISES

1. Extract more features from the date features (week of the month, week of the year, day half, weekend, weekday, etc.).
2. Include a Boolean transformation of the features in your dataset like that done in Chapter 4. This will increase the feature set which can become beneficial while training the model.

She was hopeful that this huge pool of features would now enable the classification model to better understand the dynamics of the underlying data. She was aware of what classification was, but her curiousity to learn more made her come up with the following material on the topic.

Classification

Classification helps us decide which of the given classes a new observation will fall into. Classification comes under supervised learning where the model can only be trained once a membership labeled data is provided as an input. These membership variables are usually categorical variables which can be nominal as well as Boolean in nature.

As an example, consider a person applying for a mortgage at one of the premier banks. The bank will access his application and would like to know if that person will be subject to default in the future or not. If not, the bank will approve his application for the mortgage. This presents a prime application for classification where the bank uses past data to train the model. It will then take into account that person's age, gender, salary index, lifestyle index, and credit score to predict whether or not he will default in the future. Here the membership variable we are trying to predict is "default," and it is categorical in nature.

The following methods can be used to evaluate a classification model:

- **Accuracy:** Classifier and predictor accuracy
- **Speed:** Time to train and predict from the model
- **Robustness:** Handling missing values and noise
- **Scalability**: Efficiency in disk-related databases
- **Interpretability:** Predictions made by the model make intuitive sense

Model Evaluation Techniques

Python allows the provision of measuring classification performance with the aid of several score, loss, and utility functions. These metrics require probability estimates of confidence values, positive class, binary decision values or value within the `sample_weight` parameter (i.e., weighted contribution of each sample to the overall score). These can be divided in several ways.

Confusion Matrix

Confusion matrix counts the true negatives, false positives, false negatives, and true positives.

- True negatives is the frequency of instances in which the model correctly predicted 0 as 0.
- False negatives is the frequency of instances in which the model predicted 1 as 0.
- True positives is the frequency of instances in which the model correctly predicted 1 as 1.
- False positives is the frequency of instances in which the model predicted 0 as 1.

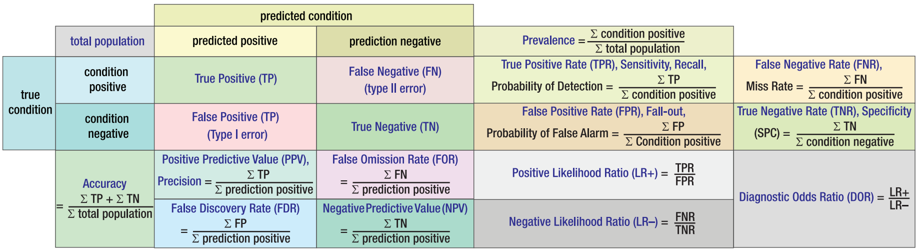


Figure 5-9. *Confusion matrix along with the respective formulas*

While doing research Dr. Judy discovered a simplified version of the Confusion matrix which shows the model evaluation as a visual representation.

Binary Classification: Receiver Operating Characteristic

A good way to understand the mechanics behind ROC (Receiver Operating Characteristic) curves is by means of a Confusion matrix. ROC curves visually plot the performance of a binary classifier as the discrimination threshold is varied. Keep in mind the following terms:

- **True Positive Rate (TPR):** that is, the sensitivity, recall, or probability of detection
- **False Positive Rate (FPR):** that is, 1 - specificity, true negative rate, or probability of false alarm

Consider, for example, the same dataset on which the model will be trained to predict the people who are highly subject to default in the future. We split the data into 70/30, which means that the model will be trained on 70% of the observations and tested on 30% of the observations. After having used the model to predict on 30% of the observations we ended up with the chart in Table 5-4.

Table 5-4. *Frequency of Defaults from Actual Data and the Predicted One*

	Will default	Will not default
Actual	30	70
Correctly predicted	19	36

TPR in this case will be 19/30 ~ 63%. Whereas, FPR in this case will be 36/70 ~ 51%

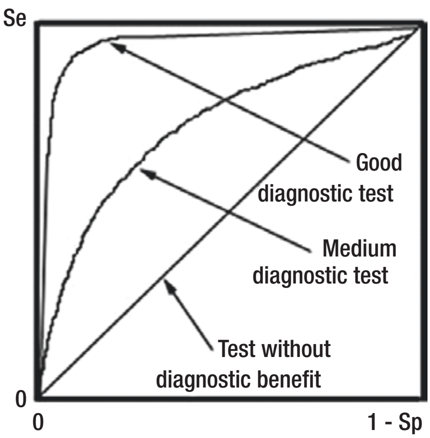


Figure 5-10. *ROC curves and relative comparison among each of them*

The ROC curve plots the fraction of true positives out of the positives (i.e., TPR) versus the fraction of false positives out of the negatives (i.e., FPR). Or, in other terms, we can say a ROC curve is the sensitivity as a function of fallout. It plots a cumulative distribution function that helps select possibly optimal models.

Accuracy on the curve is represented by Area under the Curve (AUC). Values range between 0.0 and 1.0.

Dr. Judy decided to define a method that would take the actual data and predict the goodness of the model. For that reason she wrote the code snippet in Listing 5-24.

Listing 5-24. Declaring a Function to Detect Model's Accuracy by Applying Methods Learned Previously

```
def model_performance(model_name, X_train, y_train, y_test, Y_pred):

    print 'Model name: %s'%model_name
    print 'Test accuracy (Accuracy Score): %f'%metrics.accuracy_
    score(y_test, Y_pred)
    print 'Test accuracy (ROC AUC Score): %f'%metrics.roc_auc_
    score(y_test, Y_pred)
    print 'Train accuracy: %f'%clf.score(X_train, y_train)

    fpr, tpr, thresholds = metrics.precision_recall_curve(y_test, Y_pred)
    print 'Area Under the Precision-Recall Curve: %f'%metrics.auc(fpr, tpr)

    false_positive_rate, true_positive_rate, thresholds = metrics.roc_
    curve(y_test, Y_pred)
    roc_auc = metrics.auc(false_positive_rate, true_positive_rate)

    plt.title('Receiver Operating Characteristic')
    plt.plot(false_positive_rate, true_positive_rate, 'b',
    label='AUC = %0.2f'% roc_auc)
    plt.legend(loc='lower right')
    plt.plot([0,1],[0,1],'r--')
    plt.xlim([-0.1,1.2])
    plt.ylim([-0.1,1.2])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

In order to cross-validate the model, Dr. Judy knew that she would have to split the data into training and testing data.

Ensuring Cross-Validation by Splitting the Dataset

Dr. Judy planned to train the model from the training dataset and then use that model to predict the values in the test dataset. She planned on doing so to determine the goodness of the model. She wrote the code for the training/testing split in Listing 5-25.

Listing 5-25. Declaring Features for Model Training and Splitting Data into Training and Testing Sets

```
features_of_choice = [u'Age', u'Gender', 'DayOfTheWeek', 'Diabetes',
'Alcoolism', 'HiperTension',
                        'Smokes', 'Scholarship', 'Tuberculosis',
                        'Sms_Reminder',
                        'AwaitingTime', 'AppointmentRegistration_year',
                        'AppointmentRegistration_month',
                        'AppointmentRegistration_day', 'ApointmentData_
                        year', 'ApointmentData_month',
                        'ApointmentData_day', 'AppointmentRegistration_
                        hour', 'AppointmentRegistration_min',
                        'AppointmentRegistration_sec']

x = np.array(data[features_of_choice])
y = np.array(data['Status'])

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=1)
```

Dr. Judy explained her logic in Listing 5-25. She wrote the code to take a 70-30 split whereby 70% of the observations fall into training and 30% of observations fall into the test dataset.

Before applying the classification model she thought it better to recall the reason for doing classification in the first place. With the aid of the exploratory analysis she had figured out the reason for recurring losses. Then she decided to go a further mile to find a remedy for this problem. She decided to use classification as an aid to predict if a patient will show up on his/her appointment day or not. This would enable management to do either of the following:

- Scale down the human resources (i.e., staff and doctors) to cut costs
- Determine the reasons for patients not showing up and find fixes for that problem

While looking at classification, the only visual representation Dr. Judy could think of was a tree. As in the case of Status, the tree will have one parent node and two child nodes (i.e., show-ups and no-shows). While searching for classification models, she found a visual representation, Decision Tree Classification.

Decision Tree Classification

Decision trees form a tree in a hierarchical fashion with each node having a decision boundary to proceed downward. The tree stops branching out at the level where there are no more splits possible. Interior nodes represent input variables having edges to each of the children. Children split the values from the input variable. They do that by partitioning the data at each level with nodes branching out to children. This behavior is known as recursive partitioning. Decision trees are easy to interpret and time efficient, and hence they can work well with large datasets. Decision trees can also handle both numerical and categorical data, that is, regression in case of numerical and classification in case of categorical data. However, the accuracy of decision trees is not as good as that produced by other machine learning classification algorithms.

Moreover, decision trees generalize highly to the training dataset and thus are highly susceptible to overfitting. A decision tree aims to partition the data so that each of the partitioned instances has similar/homogeneous values. ID3 algorithms are used to calculate the homogeneity of a sample, and if it is completely homogeneous it translates into an entropy of 0 and into a value of 1 or vice versa. A decision tree is a form of a parametric supervised learning method, and by parametric we mean that it can be applied to any data regardless of its underlying distribution.

Dr. Judy wrote the code in Listing 5-26 to train the decision tree classification model on a training dataset. She used the test-train split variables (i.e., y_test, y_train, x_test, x_train) from Listing 5-25.

Listing 5-26. Training the Model by Applying Decision Tree Classifier

```
clf = DecisionTreeClassifier()
clf.fit(x_train, y_train)
```

Output

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
presort=False, random_state=None, splitter='best')
```

Dr. Judy pointed out that because no configuration parameters were passed to the decision tree classifier, it took the default values of configuration parameters. The next step was to apply the trained model on a testing dataset to find the predicted labels of Status. She then aimed to compare the predicted labels to the original label of Status to calculate accuracy of the model.

Listing 5-27. Finding Accuracy of Decision Tree Classifier

```
y_pred = clf.predict(x_test)
model_performance('Decision tree classifier', x_train, y_train, y_test,
y_pred)
```

Output

```
Model name: Decision tree classifier
Test accuracy (Accuracy Score):           0.589040
Test accuracy (ROC AUC Score):            0.523605
Train accuracy:                           0.999952
Area Under the Precision-Recall Curve:    0.112530
```

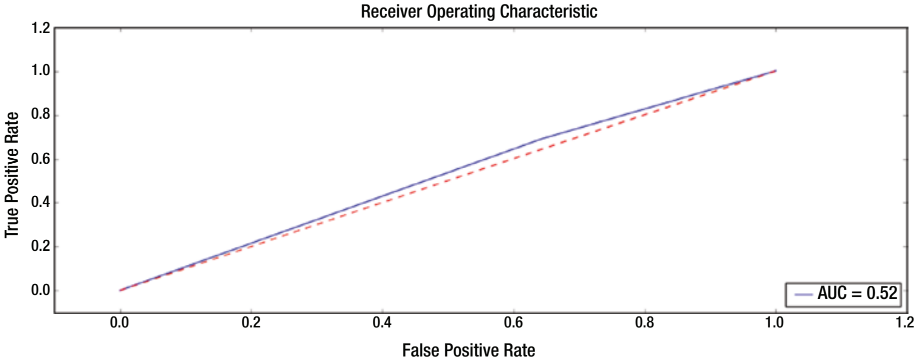


Figure 5-11. *ROC curve for decision tree model*

Dr. Judy explained that the model was trained on x_train, and y_train. The trained model (i.e., clf) was then used to predict the labels of the test dataset (i.e., y_pred). Her output of Listing 5-27 presented the perfect representation of overfitting as the train accuracy was approaching 1. Test data accuracy came out to be subpar as the value was 0.5, which was distant from 1 (i.e., the perfect score). Dr. Judy was disappointed by the results but didn't lose hope as she had many more classification techniques to experiment with.

Dr. Judy was curious to know if techniques exist to enable non-linear learning for classification models. The search for such a technique brought her to the concept of kernel approximation.

Kernel Approximation

Kernel approximation performs non-linear transformations of the input to make it suitable for linear classification and other algorithms. This is better than kernel trick as it can significantly reduce the cost of learning with very large datasets. The combination of kernel map approximations with SGD classifier can make non-linear learning on large datasets possible.

SGD Classifier

In SGD classifier, the gradient of the loss is updated one sample at a time, and the model is updated with respect to the learning rate. In order for the model to give the best results on default learning rate, the data should have zero mean and unit variance.

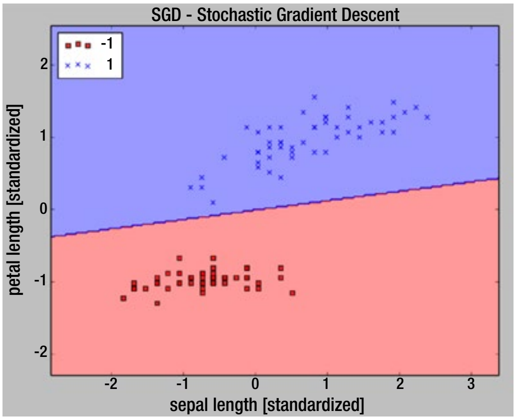


Figure 5-12. *SGD classification results on Iris dataset*

This algorithm works best with data which is represented as sparse arrays of floating point values for features within the dataset. Loss parameter controls the model it fits, which is Linear Support Vector Machine (SVM) by default.

Without waiting any longer, Dr. Judy wrote the code snippet in Listing 5-28 to train kernel approximation along with the SGD classifier model on the training dataset.

Listing 5-28. Training the Model by Applying Kernel Approximation with SGD Classifier

```
rbf_feature = kernel_approximation.RBFSampler(gamma=1, random_state=1)
X_train = rbf_feature.fit_transform(x_train)

clf = SGDClassifier()
clf.fit(X_train, y_train)
```

Output

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
eta0=0.0, fit_intercept=True, l1_ratio=0.15,
learning_rate='optimal', loss='hinge', n_iter=5, n_jobs=1,
penalty='l2', power_t=0.5, random_state=None, shuffle=True,
verbose=0, warm_start=False)
```

Dr. Judy fitted the model twice in Listing 5-28. The reason behind this was that at first she sampled from the data by fitting and transforming by means of the RBF Sampler. Per Dr. Judy, a RBF Sampler approximates the feature map of a RBF kernel prior to applying a linear algorithm. She also pointed out that fit within `fit_tranform` performed the Monte Carlo sampling whereas the transform term performed the mapping of the data.

She then wrote the code in Listing 5-29 to apply the trained model to the testing dataset and then used it to find the predicted labels along with the model's accuracy.

Listing 5-29. Finding Accuracy of Kernel Approximation with SGD Classifier

```
X_test = rbf_feature.fit_transform(x_test)
Y_pred = clf.predict(X_test)
model_performance('Kernel approximation', X_train, y_train, y_test, Y_pred)
```

Output

```
Model name: Kernel approximation
Test accuracy (Accuracy Score):          0.695619
Test accuracy (ROC AUC Score):           0.500000
Train accuracy:                          0.698398
Area Under the Precision-Recall Curve:   0.152191
```

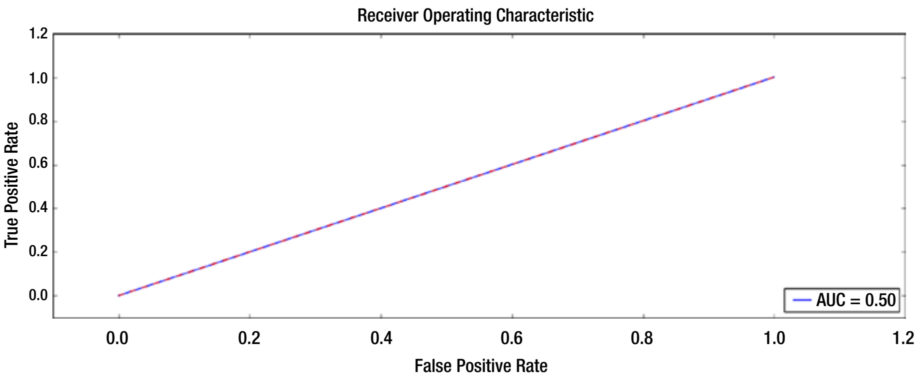


Figure 5-13. *ROC curve for kernel approximation with SGD classifier*

Training accuracy dropped relative to the decision tree classification, which means that the model didn't highly generalize on the training dataset, a good notion indeed. Also, while looking at Figure 5-13, Dr. Judy noticed that test data accuracy showed a massive improvement by shooting up from 0.59 to 0.7. Despite showing improvements in the above-mentioned aspects, the value of AUC for ROC dropped down, which surprised her a lot. After searching online for hours she remained clueless. Hence she decided to meet her friend Zoe and ask for her feedback. Once Zoe heard the problem, she was able to provide the following explanation:

> *Well the accuracy only tends to measure performance of the algorithm in detecting the true positives. Where it falls short is to take into account the False Positives and False Negatives. These False levels brought down the AUC down in spite of a higher accuracy.*

Zoe's help enabled Dr. Judy to see clearly through the confusion. However, she wasn't yet satisfied with the accuracy of the models she had explored so far. She was wondering whether techniques exist that can enable combining the power of more than one model to build a model that can do predictions with top-notch accuracy. After a thorough search, Dr. Judy found out that "ensemble" is the term that describes the representation she desired. She also discovered that ensemble techniques are subdivided into two types (i.e., Boosting and Bagging), which she believed were important to understand before using them to train the model.

Ensemble Methods

An ensemble method combines predictions from multiple machine learning algorithms, which result in relatively more accurate predictions than an individual model could have captured. Ensemble methods are usually divided into two variants (Bagging and Boosting).

Bagging

Bagging, also known as a bootstrap method, optimizes on minimizing the variance. It does that by generating additional data for the training dataset using combinations to produce multisets of same size as that of the original data. The application of Bagging is ideal when the model overfits and you tend to go to higher variance. This can be taken care of by taking many resamples, each overfitting, and averaging them out together. This in turn cancels some of the variance.

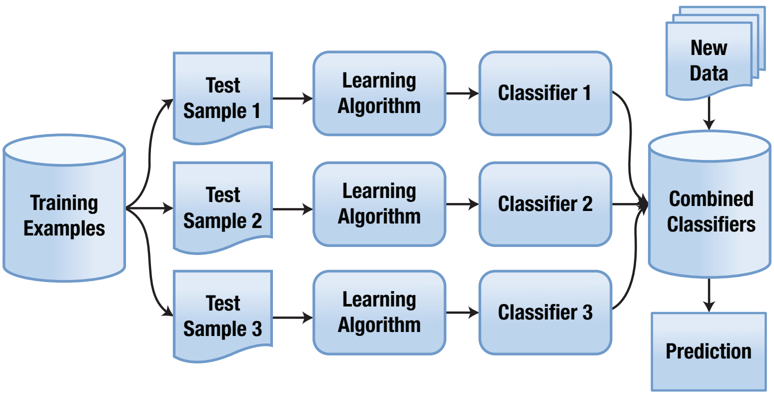


Figure 5-14. *Basic work flow of Bagging algorithms*

Decision trees are sensitive to specific data on which they are trained on. If training data is changed, the resulting decision tree can be quite different and can yield different predictions. A decision tree being a high-variance machine learning algorithm has the application of Bagging by means of the bootstrap procedure.

Consider a dataset that has 50 features and 3,000 observations. Bagging might create 500 trees with 500 random observations for 20 features in each tree. Finally it will average out the predictions for all of those 500 tree models to get the final prediction.

Boosting

Boosting defines an objective function to measure the performance of a model given a certain set of parameters. The objective function contains two parts: regularization and training loss, both of which add to one another. The training loss measures how predictive our model is on the training data. The most commonly used training loss function includes mean squared error and logistic regression. The regularization term controls the complexity of the model, which helps avoid overfitting. Boosting trees use tree ensembles because they sum together the prediction of multiple trees.

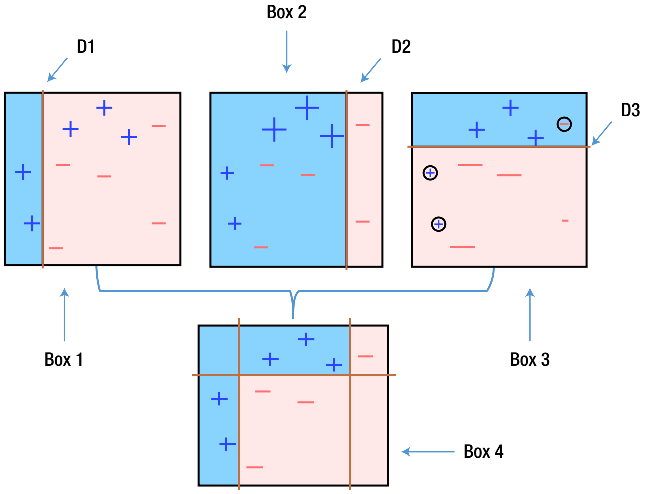


Figure 5-15. *Basic work flow of Boosting algorithms*

Dr. Judy decided to put Bagging into the application by applying the most sought Bagging technique (i.e., random forest classification).

Random Forest Classification

Random forest classification is a type of Bagging, and it is one of the most powerful machine learning algorithms available currently. In decision tree classification, different subtrees can have a lot of structural similarities which can result in prediction outputs

that are strongly correlated to each other. The random forest classifier reduces this correlation among the subtrees by limiting the features at each split point. So, instead of choosing a variable from all variables available, random forest searches for the variable that will minimize the error from a limited random sample of features. For classification, the number of variables at each split can be defined as follows:

- Symbol: m
- Formula: $\sqrt{p}$
- Where,
    a. 'm' is the optimal number of variables at each split
    b. 'p' is the number of total variables in the dataset

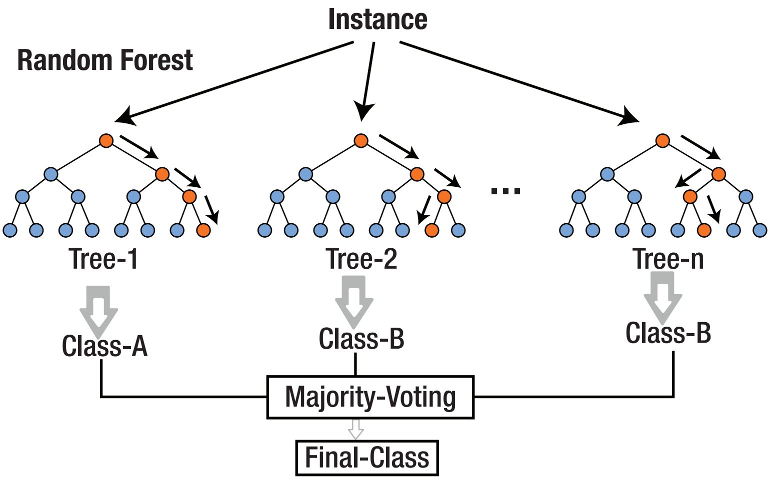


Figure 5-16. *Illustration of random forest classifier*

Random forest classifiers are fast and can work with data which is unbalanced or has missing values. She wrote the code in Listing 5-30 to apply this ensemble technique to the application.

Listing 5-30. Training RandomForest Classifier on Training Dataset

```
clf = RandomForestClassifier()
clf.fit(x_train, y_train)
```

Output

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,
```

```
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
verbose=0, warm_start=False)
```

Dr. Judy applied the random forest classifier with the default parameters configuration. She then wrote the code snippet in Listing 5-31 to evaluate the goodness of the random forest classifier model.

Listing 5-31. Finding Accuracy of the Random Forest Classifier Model

```
y_pred = clf.predict(x_test)
model_performance('Random Forest', x_train, y_train, y_test, y_pred)
```

Output

```
Model name: Random Forest
Test accuracy (Accuracy Score):          0.640874
Test accuracy (ROC AUC Score):           0.534295
Train accuracy:                          0.990071
Area Under the Precision-Recall Curve:   0.132014
```

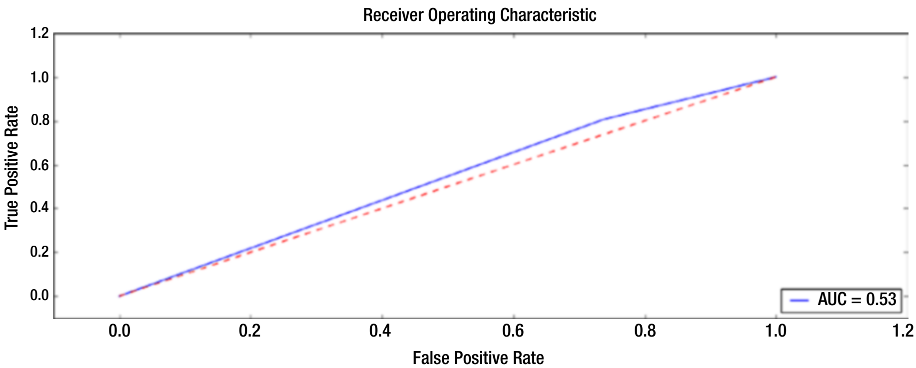


Figure 5-17. *ROC curve for random forest classifier*

Train accuracy resembled that of the decision tree classifier. It appeared that both of the tree classifier techniques overfitted the model. However, contrary to the decision tree classifier, test accuracy of the random forest classifier increased considerably. However, the test level accuracy was nowhere close to that gained from the SGD classifier with kernel approximation (i.e., 0.7 vs. 0.64).

Having applied the random forest classifier, Dr. Judy was interested to see if the boosting ensemble would yield a strong model or not. For that purpose she planned to apply gradient boosting to the train-test split obtained from Listing 5-24.

Gradient Boosting

In Boosting, the selection of samples is done by giving more and more weight to hard-to-classify observations. Gradient boosting classification produces a prediction model in the form of an ensemble of weak predictive models, usually decision trees. It generalizes the model by optimizing for the arbitrary differentiable loss function. At each stage, regression trees fit on the negative gradient of binomial or multinomial deviance loss function.

In simple terminology, the gradient boosting classifier does the following:

1. Gradient boosting builds an ensemble of trees one by one.
2. Predictions of all individual trees are summed.
3. Discrepancy between target function and current ensemble prediction (i.e., residual) is reconstructed.
4. The next tree in the ensemble should complement existing trees and minimize the residual of the ensemble.

Dr. Judy was optimistic that gradient boosting classifiers would yield the best results as they assign more weight to hard-to-classify samples, which will make the ensemble exert more efforts to classify these high-weight samples. For that purpose she wrote the code snippet in Listing 5-32 to train the model and then predict labels for the testing dataset.

Listing 5-32. Training the Model by Applying Gradient Boosting Classifier and Predicting Status Labels

```
clf = GradientBoostingClassifier(random_state=10, learning_rate=0.1,
n_estimators=200, max_depth=5, max_features=10)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
```

This time around, Dr. Judy passed in some parameters while initializing the model. She decided to go with a learning rate of 0.1, which is not too small and not too large. Other than that she decided on maximum iterations of 200 and a tree with a maximum depth of 5, and she limited the model to train itself from a maximum of ten features. Dr. Judy wrote the code snippet in Listing 5-33 to evaluate the model for goodness. For that purpose she passed the train, test, and predicted values as parameters to the 'model performance' method earlier defined in Listing 5-24.

Listing 5-33. Finding Accuracy of Gradient Boosting Classifier

```
model_performance('Gradient Boosting', x_train, y_train, y_test, y_pred)
```

Output

```
Model name: Gradient Boosting
Test accuracy (Accuracy Score):         0.700408
Test accuracy (ROC AUC Score):          0.514929
Train accuracy:                         0.707403
Area Under the Precision-Recall Curve:  0.153744
```

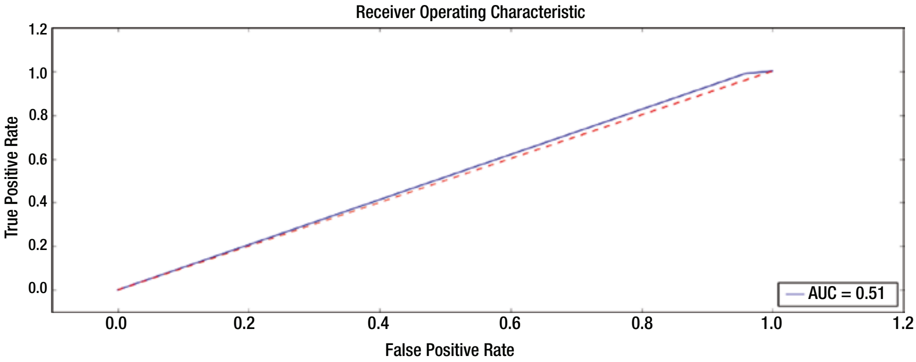


Figure 5-18. *ROC curve for gradient boosting classifier*

Dr. Judy believed that the model did a fair job of avoiding overfitting. Accuracy improved relative to the earlier applied models, with a ROC score rising over that of the kernel approximation using the SGD classifier. However, the ROC score was still lower than what came up from the decision tree classifier model.

While doing research, Dr. Judy had discovered that gradient boosting classification models also output the importance score the model gave to each feature set while training the model. She was curious to find which features outperformed others while training the model. For that purpose she wrote the code in Listing 5-34 to see features' importance scores.

Listing 5-34. Printing Features' Weight as Assigned by Gradient Boosting Classifier

```
for feature, score in zip(features_of_choice, list(clf.feature_importances_)):
        print '%s\t%f'%(feature, score)
```

Output

```
Age                                     0.143221
Gender                                  0.009629
DayOfTheWeek                            0.053643
Diabetes                                0.005889
Alcoolism                               0.010774
HiperTension                            0.006360
Smokes                                  0.011096
Scholarship                             0.010460
Tuberculosis                            0.003295
Sms_Reminder                            0.019245
AwaitingTime                            0.128393
AppointmentRegistration_year            0.023332
AppointmentRegistration_month           0.045764
AppointmentRegistration_day             0.077271
AppointmentData_year                    0.019696
```

```
AppointmentData_month                   0.066774
AppointmentData_day                     0.119192
AppointmentRegistration_hour            0.066528
AppointmentRegistration_min             0.091829
AppointmentRegistration_sec             0.087611
```

Features named Age, AwaitingTime, and AppointmentData_day were assigned the most weight by the gradient boosting classifier. Dr. Judy was thrilled to see that the model was in line with her analogy about Age and AwaitingTime while doing exploratory data analysis. Features named Gender, HiperTension, Diabetes, and Tuberculosis were given the least weight by the gradient boosting classification model.

Dr. Judy had had a productive data analysis and classification sprint. However, as the meeting was scheduled in the next half hour she couldn't proceed further, and had to present whatever findings she had gained so far. She couldn't add more to her findings unless any one of you can try some other classification alternatives for her. She had some alternatives in mind which she diligently added in the Exercises.

EXERCISES

1. Repeat gradient boosting classification but this time only consider the features it deemed important. Did AUC and ROC improve?
2. Apply grid search (check Chapter 4 for reference) to gradient boosting to fine-tune the parameters of learning rate, max_depth, etc.
3. Transform the data using PCA (check Chapter 4 for reference), which we used in the last chapter, and then apply all the models we discussed to see if we achieve improvement.
4. Recently a new type of boosting, Xgboost, has been popular among data scientists. Apply that to our dataset, optimize using grid search, and see if it performs relatively better than gradient boosting.

Dr. Judy was curious to know if there are other real-world applications to classification. For that purpose she checked the Internet and compiled some of the most compelling applications of classification to date.

Applications of Classification

Several fields of study have numerous applications for classification.

Image Classification

Deep learning has wide applications in predicting the objects represented within images. This helps in image clustering within search engines and recommendation engines in applications like Instagram.

Music Classification

Music applications like Pandora perform classification algorithms on music to recommend one that matches your preferences. The beauty of it is that you don't have to explicitly tell Pandora your preference as it will learn on its own.

E-mail Spam Filtering

E-mail services such as Gmail, Outlook, Ymail, and so on have deployed algorithms to classify spam e-mails compared to legitimate e-mails. The verdict then decides which e-mails to dump into the Spam folder.

Insurance

Insurance companies receive a huge amount of insurance claims regarding damages. Insurance companies thus have to invest a lot of time and human resources to investigate the matter and come up with the final verdict. The final verdict is right in many of the cases but wrong in some of the cases. Hence, recently efforts have been made to deploy classification models in insurance companies to discriminate fraud claim applications from the legitimate ones.

Dr. Judy summed up her analysis by recalling the initially laid objectives which were to determine reasons for recurring losses and methods to detect the occurrence of losses in the future. She started off by performing univariate and multivariate analysis of the data to determine the areas of data treatment. The exploratory analysis also enabled her to see that the losses were apparently a result of roughly 33% of patients who did not show up on their appointment day and time. After performing data wrangling, she thought it better to generate more features from the existing ones to make it easy for the model to capture the true essence of data.

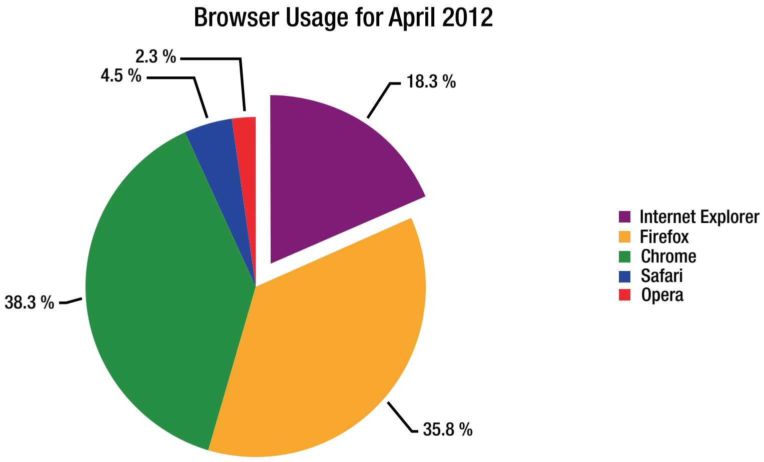
She then reviewed multiple classification model evaluation techniques and split the data into train-test splits to enable evaluation of the models. Dr. Judy started off with decision tree classifiers which showed overfitting and high test error. However, after applying a handful of classification techniques she decided on the gradient boosting classification model, which yielded relatively better accuracy. By means of the gradient boosting model she now had the power to predict in real time if a patient who had booked an appointment would or would not show up on the day of his/her appointment. She knew that she had achieved a major milestone and was now looking forward to the board meeting.

APPENDIX A

Chart types and when to use them

Pie chart

Figure illustration of pie chart

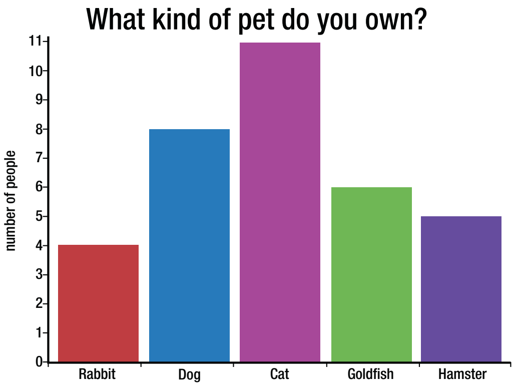


Pie chart is best to use when trying to compare parts of a whole. If is suitable for analyzing categorical variables.

D. Haroon, *Python Machine Learning Case Studies*, DOI 10.1007/978-1-4842-2823-4

Bar graph

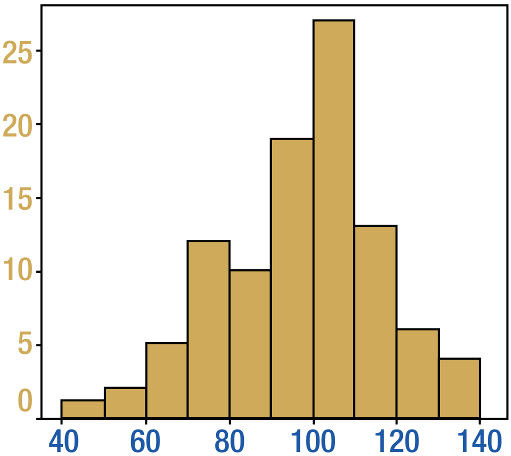
Figure illustration of a bar graph



Bar graph is used to compare things between different groups or track changes over time (i.e. when changes are large). Bar graph is suitable for categorical and interval variables.

Histogram

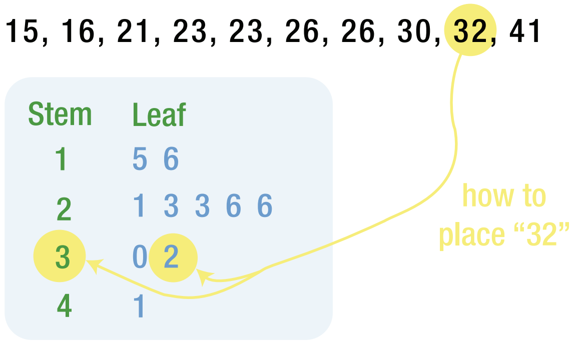
Figure illustration of a histogram



Histogram is suitable for continuous variables. It is used to plot frequency distributions with or without classes. Changing the class intervals will change the underlying distribution.

Stem and Leaf plot

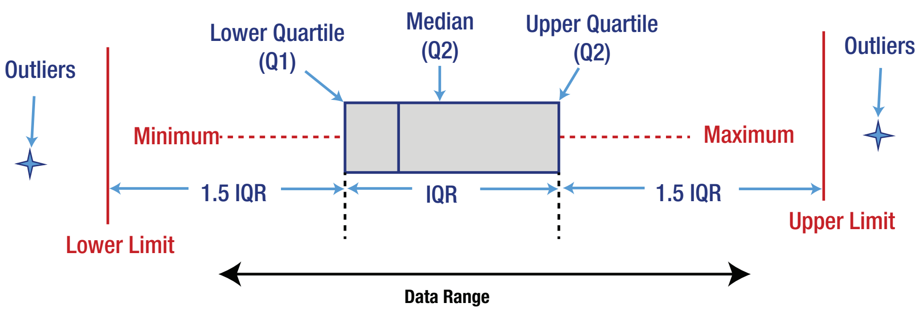
Figure illustration of a Stem and Leaf plot



Stem and Leaf plot is suitable for discrete interval variables, not that much in frequency. In other words Stem and Leaf plot can be inferred to as the transpose of a histogram. Contrary to histogram, we can reconstruct the original data from a Stem and Leaf plot.

Box plot

Figure illustration of a Box plot



Box plot is a transformed version of histogram which can help understand the median, variance and skewness of the data distribution. Line in the center is represented by the median, and lines on both ends are referred to as whiskers. Edges of the whiskers represent the first and second quartile with the difference between those referred to as an Inter Quartile Range. Points lying outside this range are considered to as outliers.

Index

D. Haroon, *Python Machine Learning Case Studies*, DOI 10.1007/978-1-4842-2823-4

D

E

F

G

H, I, J

K

L

M

N

O

P, Q

R

S

T

U, V, W, X, Y, Z